Trabajo de Fin de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo de un entorno para la gestión de respuesta inteligente en problemas de fabricación

Autor: Ángel Gutiérrez Castro
Tutor: Jesús Racero Moreno

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo de Fin de Grado
Ingeniería de las Tecnologías Industriales

Desarrollo de un entorno para la gestión de respuesta inteligente en problemas de fabricación

Autor:
Ángel Gutiérrez Castro

Tutor:
Jesús Racero Moreno
Profesor titular

Dpto. de Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: Desarrollo de un entorno para la gestión de respuesta inteligente en problemas de fabricación

Autor: Ángel Gutiérrez Castro

Tutor: Jesús Racero Moreno

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Mi más sincero agradecimiento a todas las personas que me han acompañado durante esta etapa académica que se cierra. Gracias por el apoyo y ayuda que me habéis brindado.

Agradecer también al tutor de este proyecto, Jesús Racero Moreno, la oportunidad de haberlo llevado a cabo.

Resumen

En el mundo actual, donde una empresa competitiva es aquella que se adapta a las nuevas demandas con rapidez, tiene capacidad de cambio e innovación ante nuevos requerimientos y asegura la calidad del producto, es necesario que éstas adopten metodologías que reduzcan los tiempos de planificación, diseño y construcción.

El presente proyecto abordará el diseño, desarrollo y prueba de una metodología para la simulación de sistemas de fabricación, herramienta que facilitará la construcción de sistemas productivos eficaces y eficientes.

Abstract

Nowadays, where a competitive company is one that adapts to new demands quickly, has the capacity for change and innovation in the face of new requirements and ensures product quality, it is necessary that they take methodologies that reduce planning, design and manufacturing times.

This project will address the design, development and testing of a methodology for the simulation of manufacturing systems, a tool that will facilitate the development of effective and efficient production systems.

Índice

Índice de tablas

Índice de figuras

Abreviaturas

BoM	Bill of Materials
EBoM	Engineering Bill of Materials
ERP	Enterprise Rosource Planning
FIFO	First In First Out
IBM	International Business Machines Corporation
INCOSE	International Council on Systems Engineering
JPL	Jet Propulsion Laboratory
LIFO	Last In First Out
MBE	Model Based Engineering
MBoM	Manufacturing Bill of Materials
MBSE	Model Based System Engineering
MES	Manufacturing Execution System
OMG	Object Management Group
OOSEM	Object-Oriented Systems Engineering Method
OWL	Ontology Web Language
PLM	Product Lifecycle Management
PPR	Producto-Proceso-Recurso
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
RdP	Red de Petri
RUP SE	Rational Unified Process for Systems Engineering
SA	State Analysis
SE	Systems Engineering
SysML	System Modeling Language

UML	Unified Modeling Language
V&V	Verificación y Validación
W3C	World Wide Web Consortium

1 INTRODUCCIÓN

1.1 Modelado y simulación de sistemas de fabricación

Un modelo es una aproximación, representación o idealización de determinados aspectos estructurales, comportamiento, operación u otras características de un proceso real, concepto o sistema; sirve como una abstracción del mundo real. La utilidad de los modelos es extensa; su aplicación puede abarcar gestión de información, optimización o simulación de procesos y sistemas. En el contexto de fabricación, un modelo es un artefacto digital creado para su utilización en uno o más softwares de fabricación. Un modelo podrá disponer de varias vistas con el objetivo de proporcionar diferentes funcionalidades, cada vista es una representación del sistema desde la perspectiva de una funcionalidad [1].

Una industria puede administrar su producción mediante la implementación de una gestión basada en modelado y simulación del sistema de fabricación, esto es, los modelos regirán desde el desarrollo del producto hasta el diseño de los procesos. Los modelos pueden ser usados en el sector industrial como definiciones de las cualidades de un producto y como representación del proceso. Además, permiten la compartición de datos e información entre sistemas, lo que facilitará la actuación ante nuevos requisitos, cambios en la planificación de la producción o retrasos en el aprovisionamiento de las materias primas, entre otros. Asimismo, el modelado y la simulación permiten la predicción del comportamiento, lo que posibilitará la experimentación, evaluación y comparación entre distintas oportunidades.

El entorno de modelado y simulación de sistemas de fabricación busca la automatización en la industria, donde la información sobre instalaciones y equipos esté disponible a demanda, dónde y cuándo se necesite para todas las partes interesadas. Por ello, el intercambio de modelos y datos es esencial [2].

La eficacia y eficiencia en una empresa es clave para su competitividad. Una empresa competitiva es aquella que se adecua a las nuevas demandas, tiene capacidad de cambio e innovación ante nuevos requerimientos y asegura la calidad del producto, entre otras cualidades. Por ello, una empresa basada en modelos puede ser la vía necesaria para aumentar la competitividad de una empresa. Los modelos de sistemas de fabricación suponen una reducción en el tiempo de planificación, diseño y construcción, lo que consecuentemente supone también una reducción de costes. Los líderes de la industria están de acuerdo en que la realización de modelos de sistemas de fabricación podría reducir los costes en un 50 % y reducir el tiempo de comercialización en un 45 % [3].

La representación del mundo real a un modelo puede clasificarse en tres métodos diferentes, cada método supone un lenguaje y unas condiciones. A continuación, se definen estos tres métodos:

- Sistemas dinámicos: método creado en la década de 1950 por Jay W. Forrester. El profesor Forrester definió los sistemas dinámicos como *"el estudio de la realimentación informativa de la actividad industrial para mostrar cómo la estructura organizativa, la amplificación (en las políticas) y los retrasos en el tiempo (en las decisiones y acciones) interactúan para influir en el éxito de la empresa"* [4]. Estos sistemas tratan de agrupar e integrar en un mismo flujo común todas las áreas de una industria (producción, personal, contabilidad, etc.), donde una acción en el flujo conlleva una repercusión en el mismo flujo para decisiones y acciones futuras.
- Modelado de eventos discretos: en los modelos de eventos discretos las variables cambian de valor en ciertos instantes de tiempo, es decir, los eventos son producidos por cambios de reloj. Lo contrario a un sistema discreto sería un sistema continuo, donde las variables cambian de forma continua a lo largo del tiempo.
- Modelado basado en agentes: el modelado está basado en los individuos (no en el sistema). El comportamiento del sistema será el resultado de las interacciones de muchos comportamientos individuales.

En la actualidad, los modelos de simulación se basan principalmente en software propietario o libre, que permiten representar modelos en diferentes ámbitos, ya sea en el ámbito industrial, sanitario, o bien de servicios. La proliferación de técnicas de modelado no ha sido muy extensiva; principalmente, se usan técnicas gráficas que de forma genérica se traduce en un software de simulación. Estas técnicas de modelado son poco interoperables, lo que supone un gran esfuerzo en la experimentación con datos procedentes de otros sistemas. En los últimos años han surgido nuevas metodologías, como puede ser MBSE (Modeling Based System Engineering), que permite describir mediante modelos todas las fases en el desarrollo de productos y sistemas de ingeniería.

Los modelos de análisis siempre incorporan supuestos sobre el contexto, la estructura o el comportamiento del sistema de fabricación, así como supuestos sobre la información que la parte interesada del sistema de fabricación necesita del análisis. Cuando cualesquiera de estos supuestos cambian, es posible que se requiera la reelaboración del modelo de análisis; la metodología MBSE trata de reducir estos tiempos y costes de la simulación de sistemas de fabricación.

La metodología MBSE enfatiza en la aplicación de mejores prácticas en el ámbito de la ingeniería de sistemas. Existen diversas técnicas para ello, entre las que se puede encontrar a SysML (System Modeling Language), que es considerado el estándar de la metodología MBSE. SysML es un lenguaje con semántica formal para describir el contexto y los requisitos de un sistema, sus casos de uso, estructura y comportamiento. SysML es un lenguaje de descripción de sistema muy eficaz. Sin embargo, para respaldar la toma de decisiones, se necesitan otros modelos y herramientas de análisis y optimización, por ejemplo, los modelos de eventos discretos para simular y poder investigar el comportamiento detallado de los procesos, flujos y recursos; éstos se crean y analizan utilizando paquetes de softwares comerciales (Arena o AnyLogic, entre otros) [5].

1.2 Objetivos del trabajo

El presente proyecto tendrá como objetivo el diseño, desarrollo y prueba de una metodología para la simulación de sistemas de fabricación basada en MBSE. El objetivo principal se puede desgranar en diversos objetivos parciales, que permitirán un seguimiento más preciso del objetivo perseguido.

En primer lugar, se presenta la técnica de diseño y modelado de los sistemas de fabricación: la estructura PPR (Producto-Proceso-Recurso). Esta herramienta permitirá el correcto diseño de un sistema de fabricación identificando los elementos de éste y sus requisitos, y que acabará definiendo el proceso de producción. Al final de este bloque, se describe el marco ontológico en el modelado de los sistemas de fabricación, el cual permitirá la integración de todos los elementos de un sistema de fabricación.

El siguiente bloque presenta la metodología MBSE, una metodología que cuenta con una serie de técnicas que permitirán alcanzar sistemas de fabricación de mayor calidad.

El tercer bloque se centra en estudiar el estado del arte del modelado de simulación analizando técnicas de modelado de simulación actuales. Este bloque finaliza presentado las ontologías en el modelado de simulación como herramienta para asegurar la interoperabilidad de la simulación y estudiando la inclusión del modelado de simulación en el modelado de fabricación.

Seguidamente, se describen los sistemas PLM (Product Lifecycle Management), que serán el soporte para implementar los modelos de fabricación. Por otro lado, en esta sección se presenta el concepto de interoperabilidad, habilidad que permite conseguir un modelo de simulación genérico mediante el uso de ontologías.

Por último, se encuentra un capítulo donde se realiza un análisis de entornos de simulación en AnyLogic, software de simulación que se usa en este proyecto, y un último capítulo con la prueba de la metodología en un sistema de ensamblado.

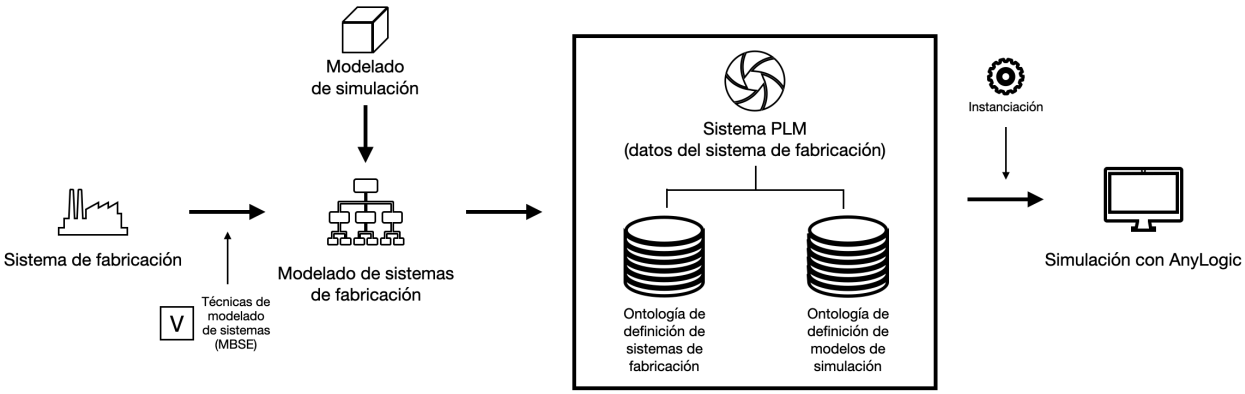


Figura 1–1. Arquitectura de la metodología del proyecto

2 MODELADO DE SISTEMAS DE FABRICACIÓN

Las actuales fábricas están expuestas a cambios constantes; el sector industrial se enfrenta a un continuo mercado cambiante. Las empresas deben lidiar con una alta tasa de reemplazo de productos y servicios, así como con la existencia de cambios significativos en los procesos comerciales y tecnológicos, instalaciones de planta, y en los recursos humanos y técnicos. Por ello, las empresas se enfrentan a un nivel extremo de competitividad, y resulta fundamental garantizar esta competitividad. Las empresas deben adaptarse, en tiempo real, a los constantes cambios de las demandas del mercado, opciones tecnológicas y regulaciones. Para alcanzar esta competitividad, los sistemas de fabricación deben ser flexibles, escalables y estar basadas en conocimiento para poder adaptarse y funcionar de manera eficaz [6].

2.1 Estructura PPR

El modelado de sistemas de fabricación resulta ser una herramienta necesaria para el diseño de éstos, la cual permitirá su adaptación a los cambios de una forma eficaz y eficiente. En este sentido, el modelado permite que los conocimientos sean reutilizados, haciendo que los tiempos de diseño se vean reducidos. No obstante, el modelado puede traer consigo el efecto contrario y dificultar el diseño de los sistemas de fabricación si no son definidos de forma precisa.

Para el correcto diseño de un sistema de fabricación ha de tenerse en consideración qué es lo que contiene y cuáles son sus requisitos. Así, la creación de una lista de materiales (BoM, por sus siglas en inglés) es un proceso clave para conseguir un correcto diseño y una producción optimizada.

Una lista de materiales es un listado completo con el que conjunto de elementos indispensables para fabricar un determinado producto. La lista de materiales incluye las materias primas, piezas, componentes, subcomponentes, si son precisos, y herramientas necesarias. La lista de materiales también incluye la relación cantidades entre los elementos que conforma el producto. Por lo general, la lista de materiales interviene en las etapas de diseño, producción y ensamblaje de un producto.

La lista de materiales tiene una estructura jerárquica donde el producto terminado se encuentra en el nivel más alto y en los niveles inferiores las partes que lo conforman.

La estructura y orden en que se fabrica un producto también puede ser adaptado en una lista de materiales. En este caso, se habla de lista de materiales de fabricación (MBoM, por sus siglas en inglés), también conocida como *"As Planned"*. Por otro lado, se cuenta con la lista de materiales de ingeniería (EBoM, por sus siglas en inglés), también conocida como *"As Designed"*. Esta lista de materiales se crea en la fase de diseño y contiene todos los componentes que unidos forman un producto, hace referencia a cómo el producto terminado ha sido diseñado previamente.

Las listas de materiales asegurarán que todo un proceso de producción esté coordinado, pues suponen el punto de unión entre todas las partes interesadas en el ciclo de vida de un producto. Por ejemplo, contar con una EBoM rigurosa permitirá que ante el lanzamiento de un nuevo producto se garantice que los materiales y piezas que lo componen estén disponibles para su fabricación. Las listas de materiales beneficiarán a las fábricas en procesos como la planificación del aprovisionamiento de las materias primas o la detección de errores. Por otro lado, en caso de que se produjeran cambios de diseño, las listas de materiales harán que los tiempos y costes de diseño se vean reducidos, así como posibilitar una fácil adaptación al nuevo proceso productivo.

Como se indicaba en párrafos anteriores, las listas de materiales tienen una construcción jerárquica. Por ejemplo, en el caso de la producción de un conjunto ensamblado, el orden de producción sería: pieza → subensamblaje → conjunto ensamblado. Así, un producto (conjunto ensamblado) está vinculado al proceso que define su orden de construcción. Además, este proceso está vinculado a su vez a los recursos necesarios

para completar la producción. Estos tres conceptos (producto, proceso y recurso) están interrelacionados y será la definición de una estructura PPR la que ayude a describir con detalle la lista de materiales y, en consecuencia, el proceso de producción.

Principalmente, en un proceso de diseño y fabricación se identifican tres elementos: el producto, ya sea en diseño o fabricación; los procesos necesarios para la obtención del producto; y los recursos o elementos utilizados para obtenerlo. Estos tres elementos que forman una estructura PPR se definen de la siguiente forma:

- Producto. Un producto puede tratarse tanto de un elemento final, como de un elemento intermedio. El concepto de producto intermedio hace referencia a productos componentes de un ensamblado, que tendrá como resultado un nuevo producto. Un mismo concepto de producto podrá ser considerado como "producto" para su fabricante o un "producto intermedio" para quien lo integra en un ensamblaje.
- Proceso. Un proceso se define como la actividad o conjunto de actividades que representan un cambio de producto durante la producción. Los productos marcarán la necesidad de uno o más procesos: ensamblaje, mecanizado, tratamientos, transporte, etc.
- Recurso. El concepto recurso hace referencia a un software, una entidad hardware o personal implicado en un proceso.

Estos tres conceptos están relacionados entre sí. La configuración de un producto determinará la necesidad de uno o más procesos en cuestión y, a su vez, será necesario el uso de uno o varios recursos.

Definir y modelar adecuadamente los requisitos de productos, procesos y recursos resulta imprescindible para el diseño de un sistema de fabricación. No obstante, no resultará completo si los modelos no se conocen entre ellos, es decir, no están integrados. Por ejemplo, si los diseños de procesos y sistemas de recursos no están completamente integrados con los diseños de productos y viceversa, los diseños de procesos no serán plenamente conscientes de las capacidades, habilidades y competencias de los procesos y recursos existentes. En consecuencia, se induce a errores en el diseño del sistema. Por otro lado, en el caso de que se produjeran cambios en alguna de las características de un producto, proceso o recurso, la no integración de los modelos no permitirá la reutilización de éstos y, por tanto, los tiempos de desarrollo de los nuevos sistemas no se verían reducidos.

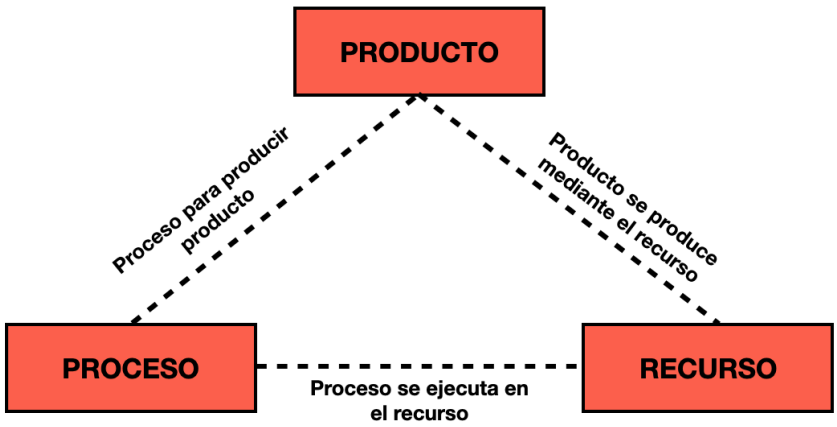


Figura 2–1. Estructura PPR

El primer paso hacia una integración de los modelos debe ser la caracterización de las relaciones a nivel semántico de todos los datos. La semántica, la cual hace referencia al significado de los datos, es necesaria para llevar a cabo la integración de los requisitos de productos, procesos y recursos. Un lenguaje semántico común actuará como eje principal e integrador de comunicación entre los conjuntos de datos de productos, procesos y recursos. Para lograr esto, se usarán y crearán estructuras ontológicas, que describirán, adaptarán y unificarán los requisitos de productos, procesos y recursos. Las ontologías son mecanismos de definición para el intercambio de conocimiento.

2.2 Ontologías

2.2.1 Introducción a las ontologías

La web semántica es un proyecto que se basa en la idea de extender la web actual con información semántica que las aplicaciones informáticas puedan procesar de manera automática. La web semántica trata de organizar y estructurar el contenido de forma que mejore la interoperabilidad entre sistemas y usuarios.

El paso de la web actual a la web semántica consiste en avanzar de una web de documentos a una web de bases de datos. Las máquinas trabajan mejor con bases de datos donde la información está muy estructurada. La web semántica propone describir los datos y sus relaciones mediante identificadores semánticos, respaldados por estándares y lenguajes.

Una de las principales herramientas propuestas para describir el conocimiento que las máquinas deben procesar son las ontologías. Las ontologías tratan de mapear el conocimiento de forma exhaustiva y rigurosa con la finalidad de facilitar la comunicación y el intercambio de información, es decir, la interoperabilidad.

Gruber definió la ontología como *"una especificación explícita de una conceptualización"* [7]. El término conceptualización hace referencia a una representación abstracta y simplificada de la realidad.

Las ontologías son estructuras semánticas que codifican de manera explícita los términos y sus relaciones entre ellos. Para ello, las ontologías están formadas por los siguientes componentes [7]:

- Clases: representan los conceptos que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- Relaciones: representan la interacción y el enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio.
- Funciones: son un tipo concreto de relación, donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- Instancias: se utilizan para representar objetos determinados de un concepto.
- Axiomas: son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología.

El W3C (World Wide Web Consortium), fundado por Tim Berners-Lee, ha propuesto un conjunto de tecnologías estándar para la representación de la información en la web semántica. A continuación, se presentan algunos de los lenguajes recomendados por el W3C.

RDF / RDF-S

RDF (Resource Description Framework) es un lenguaje para la representación de la información de la web [8]. La estructura básica de RDF es la sentencia (o "triplete") que consta de:

- Sujeto: es el recurso, es decir, todo aquello que puede ser descrito.
- Predicado: es una propiedad o atributo sobre el recurso.
- Objeto: es el valor de la propiedad.

RDF permite definir modelos de datos en forma de grafo dirigido, donde cada triplete define un nodo origen, una arista y un nodo destino.

Figura 2–2. Grafo dirigido de una sentencia RDF

RDF-S (Resource Description Framework Schema) es una extensión de RDF [9]. RDF-S proporciona mecanismos para describir grupos de recursos relacionados y relaciones entre estos recursos.

OWL

OWL (Ontology Web Language) se puede utilizar para representar explícitamente el significado de los términos que componen un vocabulario y las relaciones entre dichos términos [10]. Esta representación de

términos y sus interrelaciones es lo que se denomina ontología.

OWL incorpora nuevo vocabulario y nuevas semánticas para definir ontologías, como relaciones lógicas entre clases, cardinalidad, relaciones especiales, etc. OWL tiene más facilidades para expresar significado y semántica que XML, RDF y RDF-S.

OWL proporciona tres sublenguajes:

- OWL Lite. Compatible con aquellas aplicaciones que necesitan principalmente una jerarquía de clasificación y restricciones simples.
- OWL DL. Para aplicaciones con mayor expresividad, conservando la integridad computacional y la capacidad de decisión.
- OWL Full. Es la variante más expresiva de OWL, pero no proporciona ningún tipo de garantías computacionales. Por ejemplo, en OWL Full es posible definir entidades que sean a la vez clases, propiedades e instancias.

XML

XML (Extensible Markup Language) describe una clase de objetos de datos denominados documentos XML y describe parcialmente el comportamiento de los programas informáticos que los procesan [11].

Los documentos XML se componen de unidades de almacenamiento llamadas entidades, que contienen datos analizados o no analizados. Los datos analizados se componen de caracteres, algunos de los cuales forman datos de caracteres y otros forman marcas. El marcado codifica una descripción del diseño de almacenamiento y la estructura lógica del documento. XML proporciona un mecanismo para imponer restricciones al diseño de almacenamiento y la estructura lógica.

La representación más común de un OWL se realiza mediante un esquema RDF, que es un fichero XML.

2.2.2 Ontologías en el modelado de los sistemas de fabricación

Como anteriormente se mencionaba, en base a las estructuras PPR, definir y modelar adecuadamente los requisitos de productos, procesos y recursos resulta imprescindible para el diseño de un sistema de fabricación.

En los sistemas de fabricación están involucrados distintos tipos de elementos, así como diversas partes, y todos ellos deben funcionar de manera adecuada y coordinada. Además, se debe perseguir la idea de poder reutilizar los diseños y modelos de construcción. Por ejemplo, si hubiera un cambio de requisitos en los sistemas de productos o procesos, los sistemas de recursos habrán de reconfigurarse para cumplir con los nuevos requisitos, y esto debería ser una tarea ágil si la reutilización de diseños y modelos conocidos es posible. Esto se logra mediante un lenguaje semántico común de alto nivel que actúa como columna vertebral de comunicación entre los conjuntos de datos de productos, procesos y recursos. Los conjuntos se basan en modelos y métodos necesarios para permitir la convergencia de significados a lo largo del ciclo de vida del desarrollo de sistemas de fabricación [12]. Para ello, se crearán ontologías.

Los conceptos relacionados con productos, procesos y recursos se describirán mediante una ontología para cada uno de ellos. A continuación, se definen las ontologías de producto, proceso y recurso.

2.2.2.1 Ontología de producto

La ontología de producto es considerada la base para el diseño de sistemas de fabricación. Ésta describe los componentes y la estructura del producto para establecer una lógica adecuada para el diseño del producto. Los productos pueden hacer referencia a una pieza íntegra, un subensamblaje o un accesorio necesario para unir dos componentes.

Para modelar un producto, en su fase de diseño, se deben de tener en cuenta aspectos como la función, el comportamiento, la estructura, la geometría, las características de la materia prima o las tolerancias. Además, el concepto de producto debe contemplar los elementos intrínsecos del producto, es decir, si se habla de un producto ensamblado, habrá de contener todos sus componentes. Por otro lado, el concepto de producto también abarcará otras características de estos elementos como son el material, el color, la descripción geométrica, su identificación o sus relaciones con otros elementos.

2.2.2.2 Ontología de proceso

La ontología de proceso describe los procedimientos para la realización de productos. Dentro del diseño de los sistemas de fabricación, esta ontología supondrá la combinación entre el sistema productivo y los requisitos de diseño de los productos. Los requisitos de los productos se convertirán en uno o más procesos posibles, proporcionando así la base del sistema productivo requerido.

Los procesos se componen de varias actividades o tareas de nivel inferior. Estas actividades de nivel inferior son modeladas de manera individual para después establecer las agrupaciones adecuadas. De esta forma, cuando se tuviera una nueva pieza, podrán seleccionarse y, en consecuencia, reutilizarse aquellos procesos comunes. Este sentido, el tiempo de diseño se verá reducido.

2.2.2.3 Ontología de recurso

La ontología de recurso tiene tres elementos principales: máquina, recursos humanos y *layout*.

Un sistema involucra varias máquinas con diferentes parámetros de entrada, tamaño, productividad, etc. Las bases de datos y softwares de las distintas fábricas involucradas en la fabricación de un producto resultarán ser completamente diferentes. Por ello, el modelo semántico ayudará a la integración de los diferentes sistemas industriales.

Los recursos humanos describen información como requerimientos técnicos, tipo de trabajo y características del puesto de trabajo (empleado requerido). Para conseguir el empleado adecuado, la ontología de recursos humanos establecerá la relación con la ontología de proceso y la ontología de máquinas.

El concepto *layout* describe varios tipos de configuración del *layout*, del espacio de fabricación, una vez seleccionadas las máquinas y los recursos humanos.

2.2.2.4 Integración de las ontologías

Los diferentes modelos ontológicos pueden ser fusionados para formar una ontología de nivel superior única. Para ello, dado que existirán datos que pertenecerán a diferentes dominios, será necesario la definición de reglas que vinculen los diferentes dominios de datos para conseguir un modelo acoplado.

En caso de la estructura PPR, la base para que los diferentes modelos ontológicos sean acoplados resulta encontrarse en la siguiente premisa: las capacidades de los procesos y recursos coinciden con los requisitos de los productos. Las capacidades de los procesos y recursos son denominadas atributos, que son los que describen las características de los elementos específicos. Las relaciones serán las encargadas de vincular los atributos de procesos y recursos a requisitos de productos. Procesos y recursos también resultarán estar relacionados y vinculados, pues los procesos habrán de ejecutarse mediante recursos. Se encontrarán relaciones como las siguientes: “ProductoRealizadoPorProcesos”, “ProductoNecesitaOperacionEnsamblado”, “ProductoTieneComponente”, “ProcesoTieneActividades”, “ProcesoEjecutadoPorMaquina”, etc.

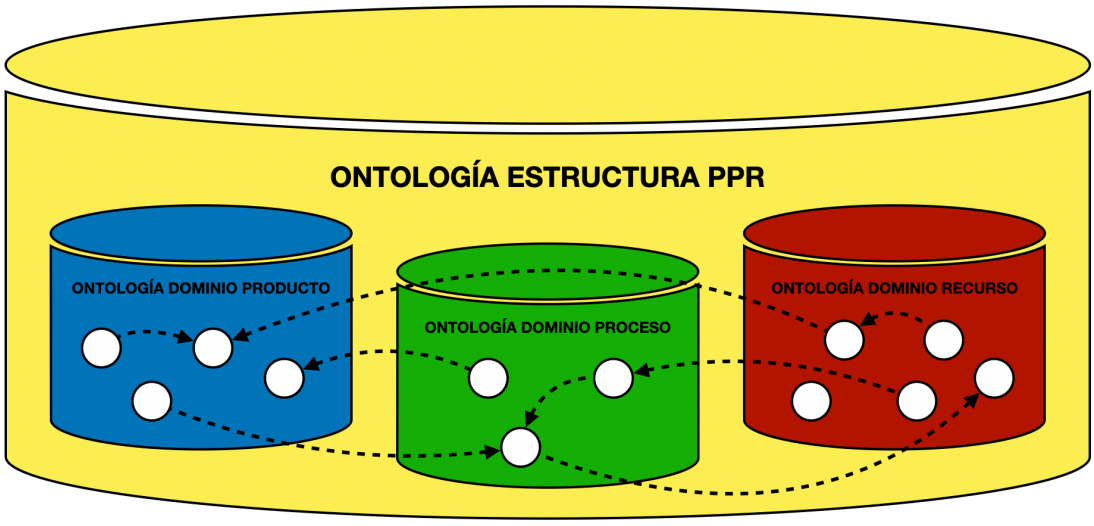


Figura 2–3. Representación ontología PPR con sus dominios acoplados

3 MODEL BASED SYSTEM ENGINEERING

La industria está repleta de grandes y complejos sistemas de fabricación. Estos sistemas de fabricación están formados de múltiples componentes (personal, software, hardware, información, procesos, instalaciones, etc.) que deben trabajar por y para unos mismos objetivos. Para ello, es necesario que estos componentes interactúen, cooperen y se adapten entre ellos.

La ingeniería de sistemas permite la creación satisfactoria de sistemas complejos desde un punto de vista disciplinar, pues la responsabilidad de la ingeniería de sistemas es la integración sistemas donde están involucrados distintos componentes heterogéneos.

Ya se ha mencionado anteriormente que el modelado de los sistemas de fabricación se hace necesario para el desarrollo eficaz y eficiente de éstos. Un modelo resulta ser una versión simplificada de un sistema real que elimina los elementos innecesarios para una determinada perspectiva, facilitando la comprensión, la toma de decisiones, la comunicación, la explicación y validación de comportamientos.

El enfoque basado en modelos aplicado a la ingeniería da lugar a la ingeniería de modelos (MBE, por sus siglas en inglés), los cuales son usados como eje central para el desarrollo de un sistema o producto durante todo su ciclo de vida, trayendo consigo la reducción en los tiempos de diseño, el aumento de la calidad de los resultados y la facilitación de la ingeniería.

El concepto de ingeniería de modelos junto con el de ingeniería de sistemas permite la definición de la metodología *Model Based System Engineering* (MBSE), que es el empleo formalizado del modelado en la ingeniería de sistemas.

3.1 Definición de MBSE

Tradicionalmente, la ingeniería de sistemas se ha desarrollado basándose en un enfoque centrado en documentos, donde se utiliza una documentación masificada basada en texto para concentrar las especificaciones de los sistemas, la administración de éstos y el intercambio de información. Sin embargo, este enfoque presenta una serie de inconvenientes cuando se presentan sistemas complejos de cierta envergadura. En estos casos, el hecho de tener una gran cantidad de documentación conllevará a tareas arduas cuando se tenga que actualizar documentos, encontrar una información en concreto o guardar la coherencia entre documentación, entre otras dificultades. Además, la colaboración entre diferentes partes deriva en ser prácticamente nula, y más si las partes se encuentran en distintos lugares.

Cuando se percibe que existen numerosos componentes, múltiples partes interesadas, excesivos requisitos del sistema, la posibilidad de que ocurran comportamientos imprevistos o cambios en el diseño del sistema, entre otras complejidades, surge la necesidad de diseñar metódicamente los sistemas. Por ello, en contraposición con el enfoque centrado en documentos, nace el enfoque centrado en modelos.

En los enfoques centrados en documentos, los cuales usan una semántica ambigua, no existe una representación precisa de las necesidades de cada una de las partes interesadas, lo que conlleva a pasar por alto información crítica y la posibilidad de crear confusiones. Por el contrario, los enfoques basados en modelos permiten la generación de arquitecturas de forma progresiva, es decir, comenzar con componentes simples para después ir construyendo subsistemas más complejos y, en última instancia, producir sistemas a partir de los componentes realizados. Esta construcción gradual, que contempla todos los componentes necesarios de un sistema y su funcionalidad, facilita la colaboración entre las partes interesadas. Para ello, los modelos permiten el desarrollo de documentos individualizados para las diferentes partes involucradas, evitando de este modo la pérdida de información y la realización de interpretaciones incorrectas. En este sentido, se deduce que será necesario la existencia de un espacio común para una colaboración significativa, un análisis exhaustivo de las necesidades, el uso de un vocabulario común y la correcta definición de la terminología y relaciones. MBSE es

una metodología que aborda todos estos desafíos.

Los modelos no solo suponen la representación de los elementos de las distintas disciplinas de un sistema real, también describen y estructuran la información sobre el sistema y su interacción con el entorno. Además, este conjunto de características, bien integradas, complementarias y asentadas acorde a la metodología MBSE, permitirá a las distintas partes interesadas ver el sistema desde la perspectiva relevante para ellas, lo que también favorece a la comunicación entre las partes. Por otro lado, los modelos son precisos para la toma de decisiones, pues son capaces de representar los distintos supuestos y requisitos para poder estudiar el comportamiento y rendimiento del sistema.

Los modelos evolucionan a lo largo de ciclo de vida del sistema, esto es, los modelos se desarrollarán conforme el alcance del sistema, sus necesidades y características vayan progresando.

Un modelo se considera idóneo cuando responde a las preguntas asociadas con los propósitos específicos requeridos. Para ello, existen dos conceptos que están muy asociados con el modelado: verificación y validación. La verificación trata de comprobar que el modelo ha sido implementado correctamente. Se puede decir que un modelo ha sido implementado correctamente si todas las características del sistema real están bien especificadas y relacionadas acordemente. Por otra parte, la validación busca que el modelo cumpla con los comportamientos planificados y deseados.

La ingeniería siempre ha usado modelos para el desarrollo de sistemas de fabricación. Los modelos son una abstracción del mundo real que permiten plasmar las múltiples perspectivas involucradas en un sistema. Sin embargo, la complejidad de los sistemas actuales hace necesario ir más allá y es preciso un nuevo enfoque en el desarrollo de sistemas. Este nuevo enfoque se plasma con MBSE.

Wayne Wymore ya introdujo en 1993 una base matemática para MBSE en su libro *"Model-Based Systems Engineering"*. No obstante, MBSE alcanza su popularidad con la introducción de estándares de modelado como SysML. Es en el año 2007 cuando se inicia la institucionalización de la práctica de la metodología MBSE, promovida por el INCOSE (International Council on Systems Engineering) en su *International Workshop* de ese año.

MBSE coloca a los modelos en el centro del desarrollo de sistemas donde, además, toda la información relacionada con el sistema se almacena y se gestiona desde un mismo almacén de datos. Esta característica permite la interconexión de los elementos del modelo, la recuperación efectiva de información y el razonamiento sobre el sistema. Esta interconectividad también permite la propagación automática de cambios de diseño, la verificación y la identificación de errores [13].

La metodología MBSE hace frente al complejo desarrollo de los sistemas de fabricación. Entre las principales ventajas y objetivos que pretende conseguir la implementación de MBSE se encuentran los siguientes [14]:

- Mejora de las comunicaciones. El intercambio de información se realiza por medio de los modelos, permitiendo tener en consideración varias perspectivas y puntos de vista. Además, la comunicación puede ser no solo entre los ingenieros de sistemas, sino entre cualquier parte interesada, como podría ser el mismo cliente.
- Mejora de la compresión y gestión del sistema. MBSE contribuye a tener una visión global del sistema. Permite ver las interconexiones de las partes y enfatiza las interdependencias, observándose las relaciones de los componentes individuales. MBSE proporciona un modelo preciso del sistema que puede ser evaluado por consistencia, corrección e integridad, observarse desde múltiples perspectivas y analizarse fácilmente desde el punto de vista de impactar en el caso de posibles cambios.
- Trazabilidad y transparencia. MBSE sigue una metodología de desarrollo del sistema que se basa en iteraciones, de forma que exista una total trazabilidad durante todo el desarrollo. Por otro lado, las iteraciones permiten la verificación y validación de los elementos que llevaron a fracaso, marcándose de esta forma la transparencia de las decisiones de diseño. Estos conceptos, que están ligados al concepto de ingeniería concurrente, reducirán los costes y tiempos de diseño, mejorarán la productividad, reducirán riesgos e impactos, y mejorarán la confianza en el sistema.
- Consistencia. El modelo desarrollado será considerado como la única fuente de verdad del sistema. MBSE contribuye a la actualización de los datos de forma concordada para que no se llegue a conflictos entre las diferentes partes del modelo y se eviten los posibles fallos consecuentes.

Las ventajas de MBSE conducen a una mayor calidad del sistema de fabricación, debido a los enormes beneficios que conlleva su implementación y debido a la prevención de errores y modificaciones en fases más avanzadas.

3.2 Metodología MBSE

3.2.1 Conceptos básicos

Una metodología MBSE es un conjunto de procesos, métodos y herramientas relacionados entre sí, que se definen de la siguiente forma:

- Proceso: conjunto de operaciones a realizar para conseguir un objetivo.
- Método: consiste en las técnicas y procedimientos para realizar una operación o tarea. Un método también es un proceso en sí mismo, tiene una secuencia de tareas a realizar.
- Herramienta: instrumento que puede mejorar la eficiencia de la tarea. La mayoría de las herramientas que se utilizan para respaldar la ingeniería de sistemas están basadas en software.

Es importante tener claro los diferentes conceptos que forman parte de una metodología, pues el uso de una metodología MBSE u otra vendrá determinado por las capacidades, habilidades y conocimientos para desarrollarla. Por ello, el entorno es un concepto que también está ligado a la metodología MBSE. El entorno será quien posibilite (o limite) el uso de las herramientas y métodos para un proyecto. Se debe tener claro que el objetivo de la metodología MBSE es facilitar la ingeniería de sistemas, no dificultarla.

El desarrollo de sistemas complejos requiere diseños que sean fácil de entender por todas las partes interesadas, planes de ingeniería que respondan al cambio, arquitecturas fáciles de modificar, metodologías adecuadas al problema en cuestión y procesos que apoyen todas las fases del ciclo de vida del sistema. La metodología MBSE trata de respaldar el desarrollo de estos sistemas basándose en lenguajes de modelado y herramientas que faciliten la ingeniería de sistemas. El lenguaje SysML es considerado el estándar de MBSE, es la técnica de modelado más común en la metodología MBSE.

Los diagramas que componen el lenguaje SysML, entre otras cualidades, permiten expresar los conceptos del sistema; las semánticas son flexibles y expresivas, lo que hace que la representación del sistema sea concisa y no ambigua; la construcción del sistema se realiza mediante los diagramas y modelos que componen esta técnica, que harán que la construcción se realice de forma eficiente y ordenada; permite abstracciones para gestionar el tamaño y la complejidad.

3.2.2 Ciclo de vida de un proyecto

MBSE es un paradigma de proceso de ingeniería de sistemas que enfatiza la aplicación de mejores prácticas a las actividades de ingeniería de sistemas a lo largo del ciclo de vida de desarrollo de sistemas. Estas actividades de ingeniería de sistemas incluyen, entre otras: análisis de requisitos, validación y verificación, análisis funcional y asignaciones, análisis de funcionamiento y especificación de la estructura del sistema [15]. En el proceso de desarrollo de sistemas estas actividades pueden llevarse a cabo en distintas fases. Por tanto, el ciclo de vida de un proyecto puede definirse como el conjunto de fases que atraviesa un proyecto hasta ser finalizado.

Las fases por las que atraviesa un proyecto están compuestas por una secuencia de actividades para conseguir ciertos objetivos. Una fase finaliza con la realización de uno o varios entregables, es decir, los productos (materiales o no) resultantes. Habitualmente, las fases suelen tomar el nombre del de alguno de sus entregables.

A continuación, se presentan las fases del ciclo de vida de un proyecto:

1. Fase de definición. En este apartado se identifica la idea; se establecen los objetivos del proyecto de acuerdo con las necesidades y requisitos. Se trata de una fase de planificación.
2. Fase de diseño. Etapa en la que se identifican soluciones, una vez que se ha decidido acometer el proyecto. En esta fase se constituye el equipo del proyecto.
3. Fase de fabricación. El objetivo fundamental de esta fase es denotar que el producto cumple con los requisitos definidos para así alcanzar los objetivos esperados. Para ello, será necesario la fabricación

del producto y se validarán y verificarán las especificaciones definidas.

4. Fase de control. Contempla el mantenimiento en caso de errores y asegurar que el producto (o productos) desarrollados cumplen con lo esperado.

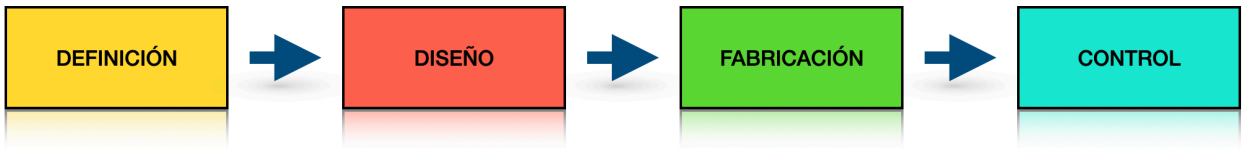


Figura 3–1. Fases del ciclo de vida de un proyecto

3.2.3 Modelos para el desarrollo del ciclo de vida de proyectos

Las diferencias entre modelos se encuentran en el alcance del ciclo, el tema y organización del proyecto, y la estructura de la sucesión de fases (lineal, con prototipo o en espiral).

Existe gran diversidad de modelos para implementar el ciclo de vida en cualquier proyecto, pero la mayoría se basan en uno de los tres modelos fundamentales: modelo en cascada, modelo en espiral, modelo en V.

3.2.3.1 Modelo en cascada

El modelo en cascada, nombrado de este modo debido a la forma de cascada que hay de una fase a otra, fue propuesto por Winston W. Royce en 1970 [16]. Este modelo tiene una estructura lineal y las principales etapas de este modelo se transforman en actividades fundamentales de desarrollo:

1. Requisitos del sistema. En esta etapa se identifican las necesidades de los usuarios.
2. Requisitos del software. Las necesidades de los usuarios son descritas como requisitos del software.
3. Análisis. En esta fase se acuerda lo que deberá hacer el producto a desarrollar. Se debe tener en cuenta que en esta fase se consensua todo lo que seguirá en las siguientes etapas y que no se podrán añadir nuevos requisitos en mitad del proceso de desarrollo.
4. Diseño de programa. Llegado a este punto, se describe la estructura interna del software. El sistema puede descomponerse y organizarse en elementos para ser elaborados por separado.
5. Codificación. Es la fase donde se programan los requisitos especificados.
6. Pruebas. En esta esta se comprueba que los componentes del sistema funcionan correctamente y cumplen con los requisitos, corrigiéndose los posibles errores que surjan.
7. Instalación y mantenimiento. La aplicación se instala en el entorno correspondiente y se comprueba que funciona correctamente. Además, se deben destinar recursos para mantener el funcionamiento del software, ya sea corrigiendo errores o mejorándolo.

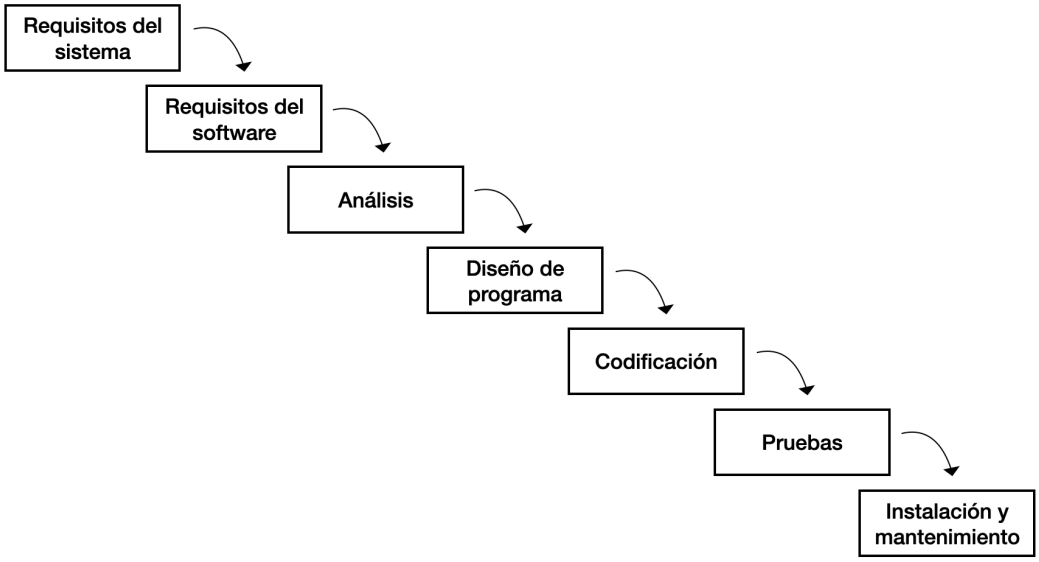


Figura 3–2. Modelo en cascada

En un segundo paso, Royce contempla una relación iterativa entre fases sucesivas, es decir, existe una iteración con los pasos anteriores y posteriores. Esta postura permite reducir a límites manejables el proceso de cambio a medida que avanza el diseño, pues se permite volver en caso de dificultades de diseño imprevistas.

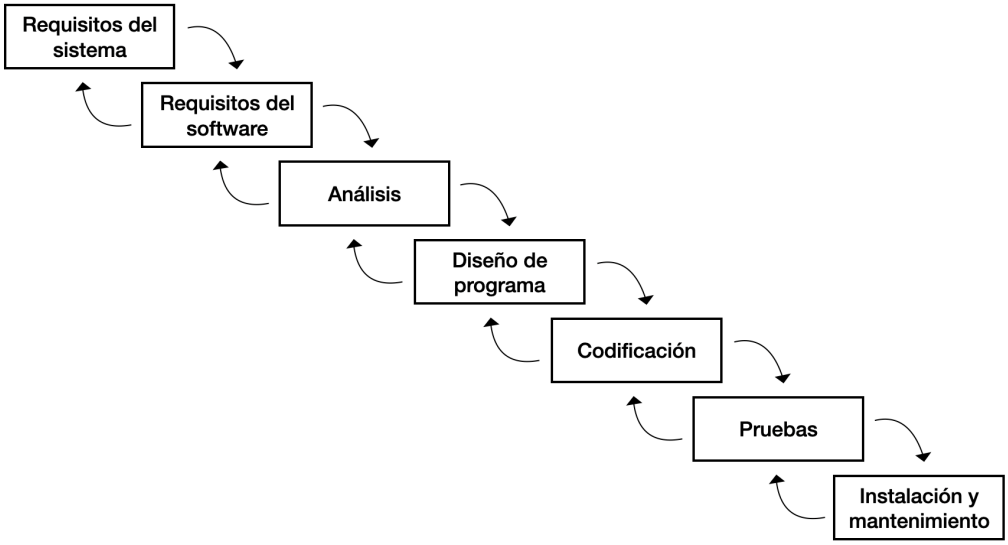


Figura 3–3. Iteración de las fases en un modelo en cascada

Por otro lado, en un siguiente paso, Royce añade la fase Diseño de programa preliminar entre las fases Requisitos del software y Análisis. Esta fase se divide en cinco componentes y mediante esta técnica el diseñador del programa asegura que el software no fallará debido a razones de almacenamiento, sincronización y flujo de datos.

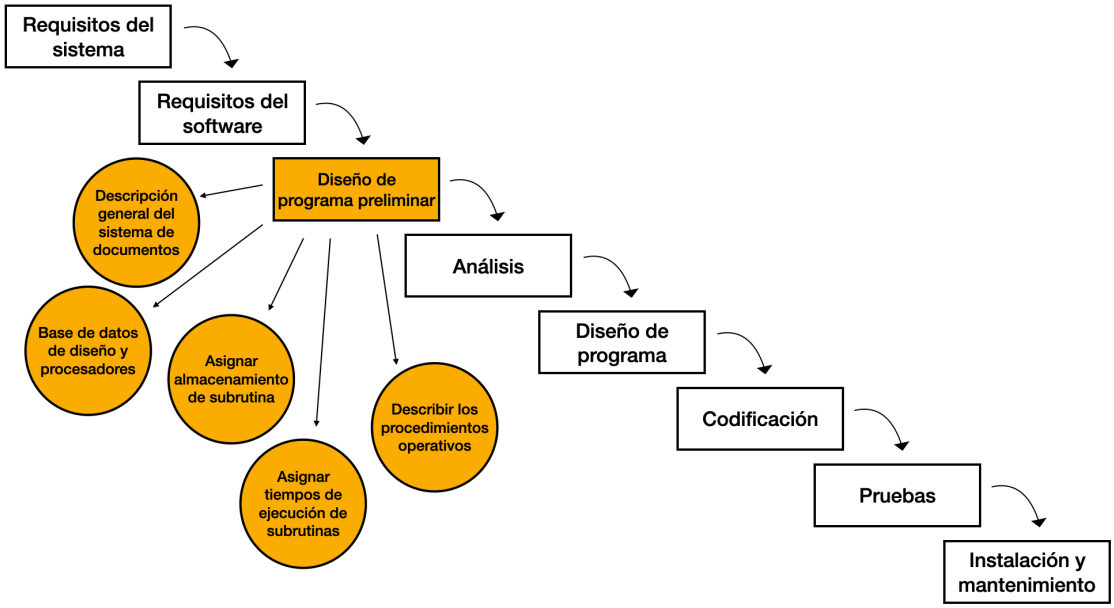


Figura 3–4. Fase Diseño de programa preliminar del modelo en cascada

La metodología de Royce incluye documentación para cada fase de desarrollo. También contempla la participación del cliente durante y después de la fase de diseño del programa, y después de las pruebas.

El modelo en cascada invierte mucho tiempo en el diseño del proyecto en las primeras fases del ciclo de vida, lo que evita problemas que serían más costosos en fases más avanzadas del desarrollo del proyecto. Sin embargo, esta ventaja puede convertirse en una desventaja cuando no se conocen o no se definen adecuadamente los requisitos, ya que este modelo no admite cambios a lo largo del proceso de desarrollo.

3.2.3.2 Modelo en espiral

El modelo en espiral, enunciado por Barry W. Boehm en 1986 [17], es un modelo de desarrollo de ciclo de vida iterativo. Este modelo consiste en una evolución basada en unos ciclos de cuatro fases cada uno, que se van realizando en forma de espiral. En cada ciclo se pasa por dichas fases, como en el modelo en cascada, pero con capacidad de evolucionar su complejidad con cada ciclo. Las cuatro fases mencionadas son las siguientes:

1. Determinar objetivos. En esta fase se determinan los objetivos, el alcance, las alternativas y limitaciones del sistema.
2. Análisis de riesgo. Se evalúa todo aquello que pueda afectar al proyecto, se definen los pasos a seguir para reducir los riesgos y se plantean alternativas. Se diseñarán los prototipos que deberán ser validados en el ciclo.
3. Implementación. En esta etapa se desarrolla y valida el software según el alcance acordado.
4. Planificación. Antes de iterar a un nuevo ciclo, se deberá revisar el avance del proyecto. Si se estima necesario continuar a un ciclo posterior de la espiral, se desarrollarán los planes para la siguiente iteración.

Con cada iteración alrededor de la espiral se crean sucesivas versiones del sistema cada vez más completas y, al final, el sistema queda totalmente diseñado.

El valor del modelo en espiral se encuentra en la evaluación del riesgo, a diferencia de otros modelos. Los riesgos son causantes de problemas en el proyecto y provocan aumentos de los costes. Por ello, la disminución de los riesgos es un hecho de significativa importancia.

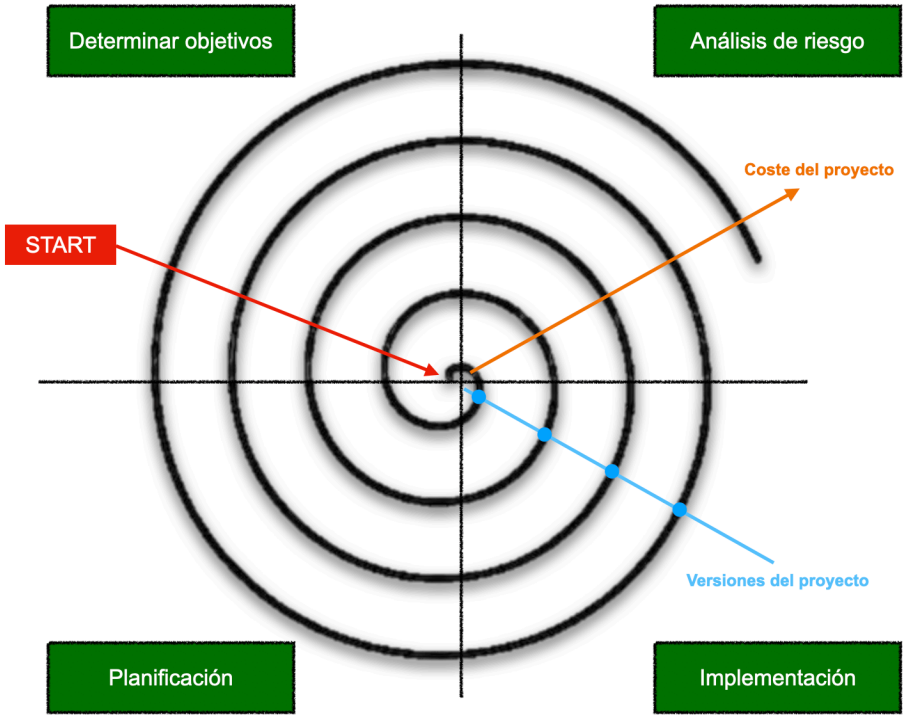


Figura 3–5. Modelo en espiral

3.2.3.3 Modelo en V

El modelo en V, también conocido como modelo V&V (verificación y validación), fue desarrollado por primera vez por Kevin Forsberg y Harold Mooz en 1992 [18]. Este modelo también constituye una representación paso a paso del ciclo de vida de un sistema, producto o servicio. El modelo trata de organizar los procesos para desarrollar sistemas y software.

El modelo en V, además de definir los procesos para desarrollar un proyecto, también detalla los procedimientos de verificación y validación que lo acompañan. Estas tareas de control de calidad están relacionadas con cada una de las fases de desarrollo. La "V" del nombre del modelo hace referencia a cómo el modelo compara las fases de desarrollo con las fases de verificación y validación. El brazo izquierdo de la "V"

contiene las tareas de desarrollo, el brazo derecho los procedimientos de control de calidad. En la unión entre los dos brazos se encuentra la implementación del proyecto.

La estructura del modelo en V es la siguiente:

- Brazo izquierdo de la V: Desarrollo del proyecto.
  Engloba las siguientes tareas:
  - Conceptos del sistema. Esta tarea tiene como objetivo desarrollar los conceptos generales del sistema.
  - Definición y análisis de los requerimientos. Trata de definir las necesidades.
  - Diseño funcional. Trata de definir las especificaciones o requisitos de funcionamiento del sistema.
  - Diseño técnico. En esta tarea se realiza el diseño detallado, siguiendo las especificaciones definidas.
- Brazo derecho de la V: Integración y verificación del proyecto.
  Contempla las siguientes actividades:
  - Pruebas unitarias. Esta tarea realiza una inspección de los distintos componentes del proyecto.
  - Integración. Se comprueba la integración de todos los pasos del proyecto.
  - Pruebas del sistema. Actividad por la cual se verifica el funcionamiento de acuerdo con las especificaciones establecidas.
  - Aceptación. Esta tarea aborda la validación del usuario y el mantenimiento del sistema.

En la misma estructura se observa también que la proximidad entre una fase del desarrollo y su fase de verificación correspondiente va decreciendo a medida que aumenta el nivel dentro de la V. La longitud de separación entre una fase de desarrollo y su homóloga de verificación intenta ser proporcional a la distancia en el tiempo.

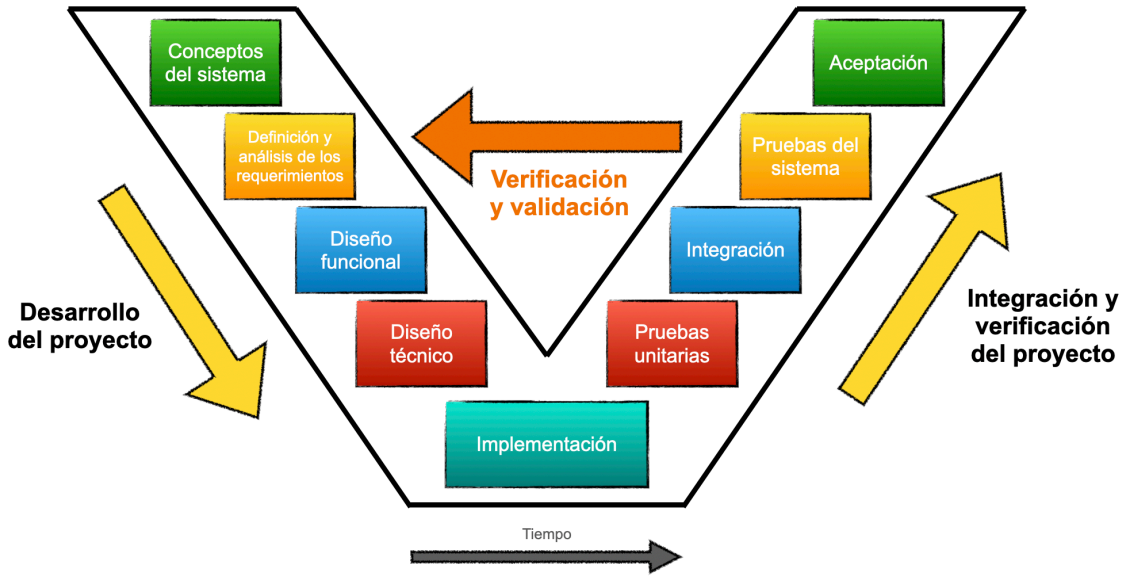


Figura 3–6. Modelo en V

Cada fase del modelo tiene que ser documentada, tanto lo analizado como los resultados de las pruebas.

3.2.4 SysML

La metodología MBSE se desarrolla mediante el uso de lenguajes de modelado. El lenguaje SysML es el estándar más común en la ingeniería de sistemas basada en modelos. SysML es un lenguaje de modelado que soporta especificación, análisis, diseño, verificación y validación de los sistemas.

SysML surge a partir de UML (Unified Modeling Language), ambos lenguajes desarrollados por OMG (Object Management Group).

UML es un lenguaje de modelado compuesto de modelos estructurales y de comportamiento, que representan las interacciones dentro de un sistema. Los diagramas estructurales tienen como objetivo documentar la arquitectura del sistema, contemplan aquellos elementos que deben estar presentes para completar un sistema.

Por otro lado, los diagramas de comportamiento describen la funcionalidad de los subsistemas y del sistema en cuestión, es decir, modelan lo que debe suceder [19].

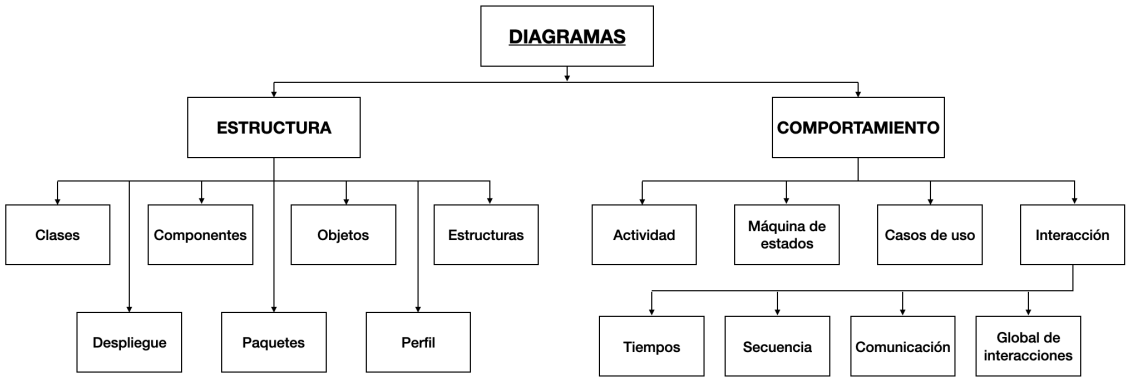


Figura 3–7. Jerarquía UML

El modelado de sistemas de ingeniería requiere la especificación, diseño, análisis, verificación y validación de los procesos, además estos incluyen hardware y software, deben tener en cuenta personas e instalaciones, la información y los procedimientos. Por ello, OMG desarrolló SysML enfocado a los sistemas de ingeniería, pues UML estaba orientado al desarrollo de software [20].

SysML incluye diagramas estructurales y de comportamiento como UML. Sin embargo, descarta aquello diagramas más orientados al desarrollo de software e incluye los diagramas de requisitos, que serán el soporte para modelar las particularidades de un sistema de ingeniería.

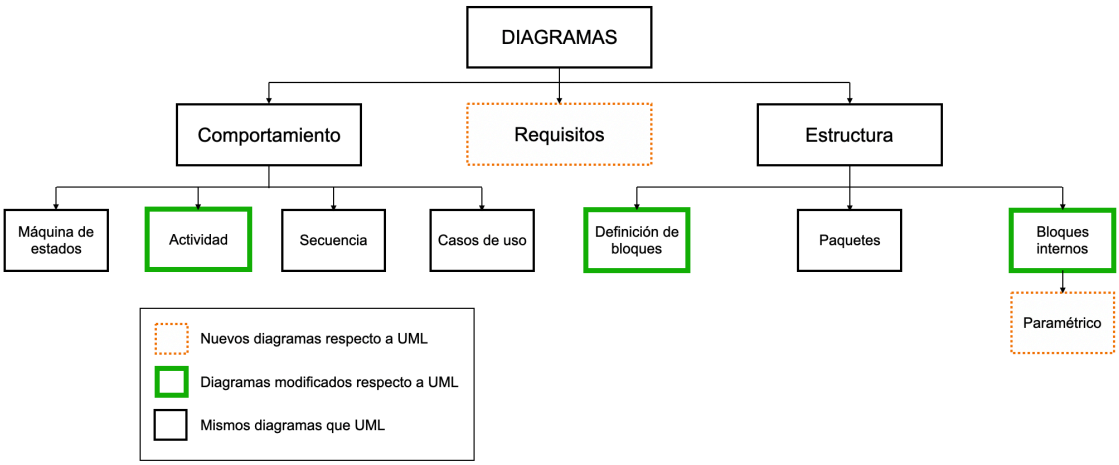


Figura 3–8. Jerarquía SysML

La jerarquía de SysML se estructura de la siguiente forma [21]:

- Diagramas de comportamiento.
  - Máquina de estados: representa el comportamiento en términos de estados y transiciones del sistema.
  - Actividad: es el encargado de molar los inputs, outputs, secuencias y condiciones para coordinar otros comportamientos.
  - Secuencia: describen las interacciones entre entidades, es decir, cómo los elementos del sistema se comunican entre sí en términos de una secuencia de mensajes y eventos.
  - Casos de uso: describen la funcionalidad y capacidad del sistema mediante la interacción entre actores.
- Diagramas estructurales.
  - Definición de bloques: representa las partes principales de un sistema como una serie de bloques, con interconexiones para representar las relaciones.
  - Paquetes: se utiliza para organizar el modelo dividiendo en agrupaciones y estableciendo dependencias entre agrupaciones.

- Bloques internos: captura la estructura interna de un sistema en términos de componentes, propiedades y relación entre las partes constituyentes. Además, este diagrama incluye un diagrama paramétrico que describe las restricciones entre las propiedades asociadas con bloques. Este nuevo diagrama de SysML se utiliza como análisis de ingeniería dentro del diseño de modelos (rendimiento, confiabilidad, etc.).

- Diagrama de requisitos. Esta incorporación, respecto a UML, muestra los requisitos del sistema y sus relaciones con otros elementos a través de modelos visuales.

SysML es un lenguaje de modelado gráfico, es decir, proporciona representaciones gráficas con una base semántica para modelar los requisitos, el comportamiento, la estructura y los parámetros del sistema.

3.2.5 Metodologías MBSE

Las metodologías MBSE más usadas en la industria actual están basadas en los tres modelos fundamentales (cascada, espiral y en V) para el desarrollo del ciclo de vida de un proyecto, así como en el uso del lenguaje de modelado SysML. A continuación, se presentan algunas de estas metodologías MBSE.

3.2.5.1 Harmony SE

Harmony SE fue desarrollada por la multinacional IBM (International Business Machines Corporation) y se usa para el desarrollo de sistemas integrados y software.

El desarrollo de esta metodología sigue el estándar del modelo en V.

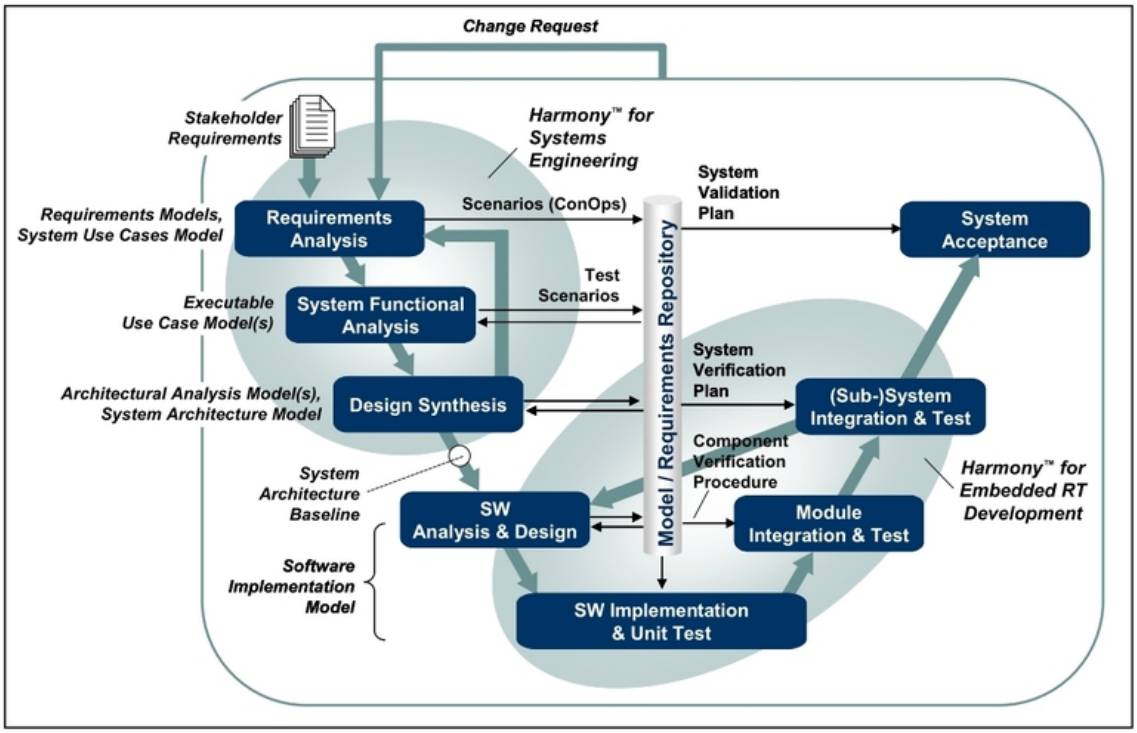


Figura 3–9. Diagrama en V del proceso Harmony [22]

Esta metodología incluye los siguientes procesos elementales:

- Análisis de requisitos. Consiste en recopilar los requisitos de las partes interesadas y traducirlos en requisitos del sistema (funcionales y no funcionales).
- Análisis funcional del sistema. Consiste en traducir los requisitos funcionales en una representación coherente de las operaciones del sistema.
- Síntesis de diseño. Proceso que consiste en el desarrollo de una arquitectura que satisfaga tanto los requisitos funcionales como las limitaciones de rendimiento.

La metodología Harmony SE se construye mediante el uso de SysML y sus diagramas [23].

3.2.5.2 OOSEM

El OOSEM (Object-Oriented Systems Engineering Method) fue desarrollado en un principio por los ingenieros de sistemas de *Lockheed Martin Corporation* y *Systems and Software Consortium*. Posteriormente, el INCOSE cuenta con un grupo de trabajo dedicado a mantener este método actualizado y desarrollarlo.

Los dos objetivos principales de este método son: (1) facilitar la integración de la ingeniería de sistemas con la ingeniería de software orientada a objetos; (2) aplicar el modelado orientado a objetos de una manera que beneficie el proceso de la ingeniería de sistemas.

OOSEM se basó originalmente en UML. Ahora, utiliza el lenguaje de modelado SysML para representar el análisis del sistema y la información del diseño. Este método contiene actividades para el análisis de necesidades y requisitos, diseño de arquitectura lógica, optimización y evaluación de alternativas, e incluye verificación y validación [24].

Las actividades que componen el método OOSEM siguen un modelo en V y se pueden aplicar de forma recursiva e iterativamente en cada nivel de la jerarquía del sistema.

3.2.5.3 RUP SE

RUP SE (Rational Unified Process for Systems Engineering) fue desarrollado por IBM como una extensión de RUP (Rational Unified Process). RUP es una metodología que profundiza en la disciplina y buenas prácticas para el desarrollo de software, RUP SE fue desarrollado para incorporar nociones de ingeniería de sistemas.

Los principales elementos de contenido de RUP son los siguientes:

- Roles. Un rol define un conjunto de habilidades, responsabilidades y competencias relacionadas.
- Productos de trabajo. Un producto de trabajo representa algo que resulta de una tarea, incluidos todos los documentos y modelos producidos mientras se trabaja en el proceso.
- Tareas. Una tarea describe una unidad de trabajo asignada a un rol que proporciona un resultado significativo.

El ciclo de vida de RUP es una implementación del modelo en espirar para el desarrollo iterativo e incremental.

Como RUP SE deriva de RUP, conserva los principios fundamentales de RUP, que se han perfeccionado y ampliado para mejorar su utilidad en la ingeniería de sistemas. Particularmente, RUP SE amplía en términos de software, hardware, personal, información y colaboración entre las partes interesadas [23].

3.2.5.4 SA

La metodología SA (State Analysis) fue desarrollada por JPL (Jet Propulsion Laboratory) y está basada en modelos y estados. Un estado se define como una representación de la condición momentánea de un sistema en evolución. Los modelos describen cómo evoluciona el estado.

SA cuenta con los siguientes principios básicos [25]:

- El control incluye todos los aspectos del funcionamiento del sistema. Debe hacerse una distinción clara entre el sistema de control y el sistema bajo control.
- Los modelos del sistema bajo control deben estar explícitamente identificados.
- Comprender el estado es fundamental para el éxito del modelado. Todo lo que necesitamos saber y todo lo que queremos hacer se puede expresar en términos de los estados del sistema bajo control.
- La manera en que los modelos informan del diseño y el funcionamiento del software deber directa, requiriendo una traducción mínima.

La metodología SA define un proceso iterativo para el descubrimiento y el modelado de estados, que permite que los modelos evolucionen según corresponda a lo largo del ciclo de vida del proyecto. SA proporciona un enfoque metódico y riguroso para las siguientes actividades principales:

- Encontrar, caracterizar, representar y documentar los estados de un sistema.
- Modelar el comportamiento de las variables de estado y sus relaciones, incluyendo información sobre las interfaces hardware.

- Capturar los objetivos de la misión en escenarios detallados.
- Seguimiento de las limitaciones del sistema y reglas de funcionamiento.
- Describir los métodos por los cuales los objetivos serán logrados.

Para cada uno de estos aspectos de diseño hay una arquitectura de control basada en el estado, conocida como *"Diamond Control"*. Esta arquitectura tiene una serie de características claves:

- El estado es explícito, representado por un conjunto de variables de estado.
- La estimación y el control están acoplados solo a través de variables de estado, estos dos conceptos deben permanecer separados.
- Los adaptadores de hardware proporcionan la única interfaz entre el sistema bajo control y el sistema de control.
- Los modelos están presentes en toda la arquitectura.
- La arquitectura valida el funcionamiento en términos de objetivos, que son restricciones sobre las variables de estado.
- La arquitectura proporciona un mapeo sencillo en el software.

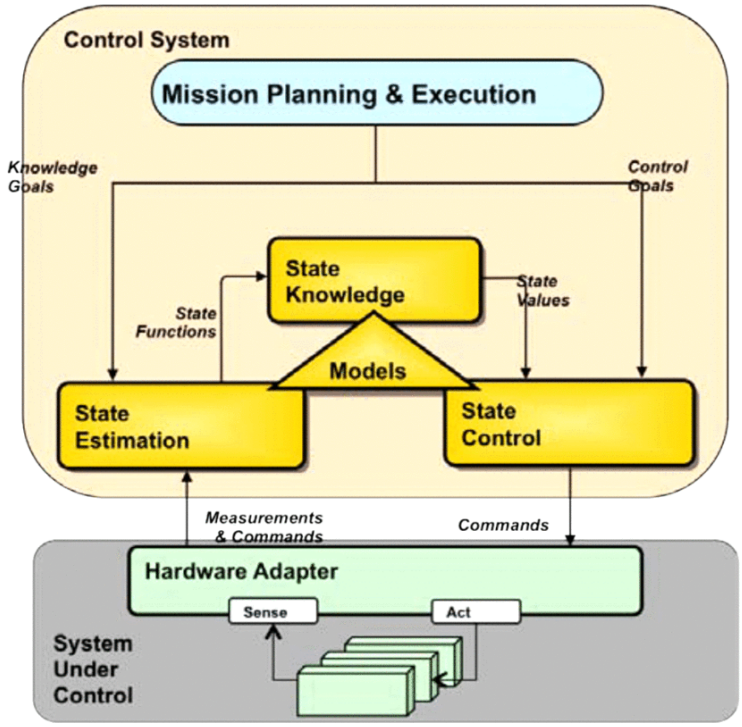


Figura 3–10. Arquitectura de control basada en el estado de SA [25]

4 HERRAMIENTAS DE MODELADO EN SIMULACIÓN

Una simulación no es más que la representación de un sistema real de acuerdo con unas reglas de modelado bien definidas y que se traslada de un estado a otro.

La simulación permite probar si se cumplen las especificaciones de diseño de un sistema utilizando experimentos virtuales en lugar de físicos, como pueden ser los prototipos. Este enfoque supone la reducción del ciclo de diseño y la minoración de su coste. Además, la simulación permite a los diseñadores de sistemas tomar decisiones de diseño inmediatas y probar más alternativas de diseño, por que se alcanzará un diseño final de mejor rendimiento. El uso de la simulación resulta ser importante para el diseño de sistemas de fabricación que están integrados por componentes de diferentes disciplinas acoplados, pues la simulación integra la dinámica de sistema.

4.1 Conceptos básicos

El modelado de sistemas de fabricación resulta ser una herramienta necesaria para un diseño eficiente de éstos. Por ello, la elección de metodologías adecuadas para el desarrollo de los diferentes modelos de simulación es primordial. Sin embargo, es poco frecuente que el desarrollo de estas herramientas de simulación esté apoyado en una metodología concreta.

Los modelos de simulación de los sistemas de ingeniería deben quedar específicamente diseñados para que puedan dar respuesta a las cuestiones de las partes interesadas, esto es, todas las propiedades del sistema deben quedar analizadas y definidas. Para ello, existen técnicas de modelado, con base en los diagramas que forman parte de la jerarquía SysML, que permiten identificar los componentes del sistema, las relaciones entre ellos y la ocurrencia de sucesos, obteniéndose de esta forma la estructura y comportamiento del sistema para después poder evaluarlo, mejorarlo y experimentar con el mismo.

Los conceptos que formarán parte de un modelo de simulación son los siguientes:

- Entidades: son los componentes cuyo comportamiento es seguido a través del sistema; se mueven, cambian de estado e interaccionan con otras entidades. Pueden ser temporales, que permanecen en el sistema solo por tiempo limitado, o permanentes, que son internas del sistema.
- Atributos: identifican el comportamiento de las entidades. Las entidades pertenecientes a una misma clase tienen los mismos atributos, pero toman diferentes valores para cada entidad concreta. Los atributos son las variables locales de cada entidad. Por otro lado, existen las variables globales, que son únicas para todo el modelo. Las variables globales no van asociadas a las entidades.
- Recursos: entidades especiales utilizadas por otras entidades para realizar una acción. Las funciones que realizan los recursos sobre las entidades son denominadas actividades.
- Colas: lugar donde esperan las entidades cuando no pueden moverse, pueden tener capacidad limitada o ilimitada.
- Sucesos: son hechos que acontecen en un determinado instante de tiempo.

El estado de un sistema queda definido por las actividades de las entidades y los valores de sus atributos. La ocurrencia de un suceso provocará el cambio en el estado del sistema. La secuencia de sucesos y los cambios de estados producidos representan el comportamiento del sistema. En este trabajo, se tratarán los sistemas de simulación de eventos discretos, es decir, los eventos que cambian el estado del sistema ocurren según unos intervalos variables de tiempo.

4.2 Técnicas de modelado

4.2.1 Grafo de eventos

Un modelo de eventos discretos puede diseñarse mediante la técnica de grafo de eventos [26]. Los grafos de eventos permiten representar estados de un sistema y cualquier tipo de relación existente entre ellos. La representación está basada en formalismos matemáticos que permiten especificar tanto la lógica, como la dinámica de un sistema de eventos discretos.

Los grafos están formados por vértices, también llamados nodos, que caracterizan a los eventos del sistema. Estos vértices cuentan con unas etiquetas que reflejan las variables de estado del sistema indicando la ocurrencia de un evento, que traerá consigo la ejecución de aquellas transacciones asociadas a ese evento.

Los vértices de un grafo están conectados mediante arcos y simbolizan las relaciones entre eventos. Los arcos se encuentran dirigidos desde un evento ocurrido hasta otro evento programado. Estos arcos tienen asociado un tiempo de retraso que indica el tiempo de recorrido del arco hasta que el evento programado suceda. Por otro lado, el evento programado tendrá lugar siempre y cuando se cumplan unas condiciones. Por ello, los arcos también cuentan con unas condiciones lógicas.

En la Figura 4–1se muestra una representación de un grafo de eventos. Este grafo de eventos se puede leer de la siguiente forma: cuando el evento *A* ocurra, se programará la transacción al evento *B*, que será ejecutado dentro de *t* unidades de tiempo, siempre y cuando la condición *c* sea validada con éxito.

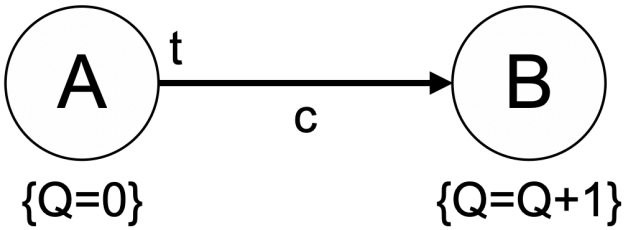


Figura 4–1. Grafo de eventos

Los grafos de eventos son utilizados comúnmente en el modelado. Esta técnica es útil para identificar las variables globales de un sistema, determinar el conjunto de eventos y eliminar los innecesarios, así como para anticipar errores lógicos.

4.2.2 Diagrama de actividades

El diagrama de actividad es una notación que ya formaba parte de UML y resulta ser uno de los diagramas modificados por SysML con respecto a UML.

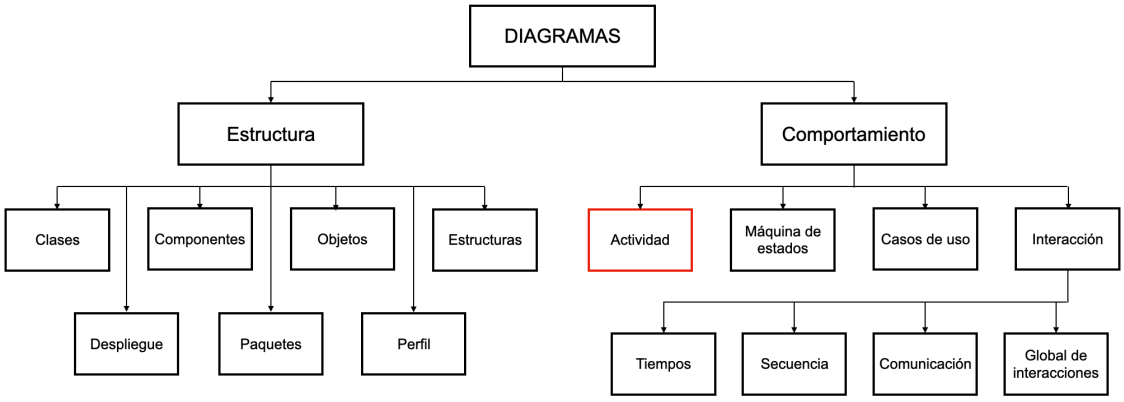


Figura 4–2. Diagrama de actividad en la jerarquía UML

Los diagramas de actividades se encargan de construir la secuencia de actividades que componen los procesos

del sistema, muestran el flujo de trabajo desde el punto de inicio hasta el punto final.

Los elementos que constituyen un diagrama de actividades son:

- Actividades. Una actividad representa una secuencia de comportamiento, es decir, un paso dentro del sistema.
- Acciones. Una acción muestra un paso dentro de una actividad.
- Transacciones. Una transacción muestra la secuencia de una acción a otra.
- Nodos inicial y final. Estos nodos indican el inicio y final de una actividad.
- Transacciones condicionales. Este tipo de transacciones condicionan el flujo a seguir, se tratan de decisiones. Las transacciones condicionales pueden servir también para la unión de varios flujos.
- Paralelismo. Algunas acciones de un proceso se realizan simultáneamente, en paralelo. Para indicar que comienzan varias acciones a la vez se usa un símbolo de comienzo de paralelismo, denominado *fork*. Cuando finalicen las acciones que se realizaban simultáneamente, se introduce un símbolo de fin de paralelismo, llamado *join*.
- Calles. Se realiza una partición de la actividad en calles cuando intervienen distintos actores.
- Flujos de objetos. Es habitual que entre acciones fluyan objetos y datos. En ese caso, se mostrarán flujos de objetos.

En la Figura 4–3 se muestra un ejemplo de un diagrama de actividad y se pueden distinguir los elementos definidos anteriormente.

Figura 4–3. Elementos en un diagrama de actividades

4.2.3 Redes de Petri

Las redes de Petri (RdP) son una herramienta de modelado gráfico y matemático, introducidas por Carl Adam Petri en el año 1962. Entre los múltiples sistemas que las RdP pueden modelar, se encuentran los sistemas de eventos discretos.

Una RdP es un grafo dirigido formado por dos tipos de nodos:

- Lugares: representados por círculos.
- Transiciones: representados mediante segmentos rectos horizontales.

Los nodos lugar y transición son conectados entre sí por flechas o arcos dirigidos. No se permiten las conexiones entre dos nodos del mismo tipo.

Los lugares tienen asociada una acción o salida. Éstos pueden presentar marcas, representadas mediante un punto en el interior del círculo. En ese caso, son considerados lugares activos. Un lugar puede tener más de una marca.

A las transiciones se les asocia eventos, que son funciones lógicas de las variables de entrada.

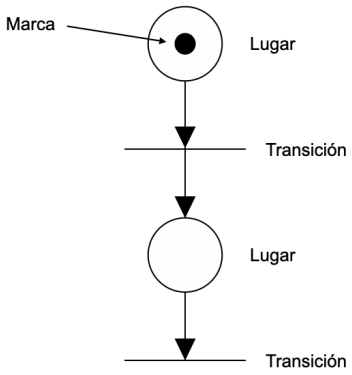


Figura 4–4. Red de Petri

Las RdP permiten el modelado de procesos independientes, paralelos o concurrentes, también sistemas donde un recurso es compartido por dos procesos, entre otras propiedades.

Las RdP resultan una herramienta muy útil para modelar sistemas complejos y de gran dimensión, su versatilidad hace que sea posible la simplificación y el modelado a distintos niveles de detalle.

4.2.4 Diagrama de clases

En la jerarquía de UML, los diagramas de estructura se encargan del modelado de la parte estática de un sistema. Dentro de los tipos de diagramas de estructura se encuentran los diagramas de clases, encargados de describir los tipos de objetos que forman un sistema y sus relaciones.

En un diagrama de clases aparecen clases relacionadas entre sí. De esta manera, las clases y las principales relaciones semánticas entre ellas pueden ser consideradas como los elementos fundamentales de estos diagramas. A continuación, se presentan el concepto de clase y las principales relaciones (asociación, agregación y herencia).

Clase

Una clase describe un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones, restricciones y semántica.

Las clases son representadas mediante un rectángulo dividido en tres secciones: nombre de la clase, lista de atributos y lista de operaciones.

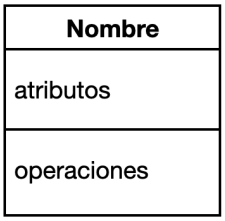


Figura 4–5. Representación de una clase

La sintaxis de los atributos es la siguiente:

visibilidad nombre: tipo = valor

Donde:

- Visibilidad: expresa si los atributos son visibles a otros objetos.
  + Visibilidad pública: visible por todos los objetos.
  # Visibilidad protegida: visible solo por el objeto y sus descendientes.
  - Visibilidad privada: visible solo por el objeto.
- Nombre: identifica el atributo.
- Tipo: indica el tipo o dominio del atributo.
- Valor: indica un valor de inicio para el atributo. Este elemento es opcional y no se da, se omitirá el signo igual.

Un atributo con alcance de clase se expresará mediante una cadena subrayada.

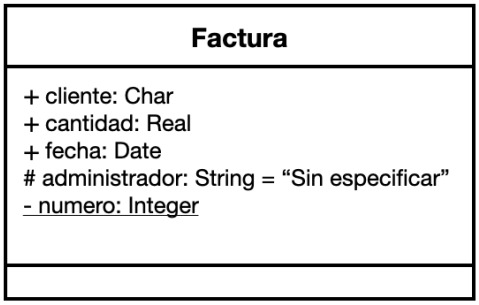


Figura 4–6. Ejemplo de definición de atributos en una clase

La sintaxis para describir una operación es la siguiente:

visibilidad nombre (lista de parámetros): tipo de retorno

Donde:

- Visibilidad, nombre: igual que para los atributos.
- Lista de parámetros: lista separada por comas de los parámetros formales. Los parámetros son definidos mediante una sintaxis igual a la de los atributos.
- Tipo de retorno: tipo de dato a devolver.

Una operación con alcance de clase se expresa mediante una cadena subrayada. Por otro lado, si la operación es abstracta se muestra en cursiva.

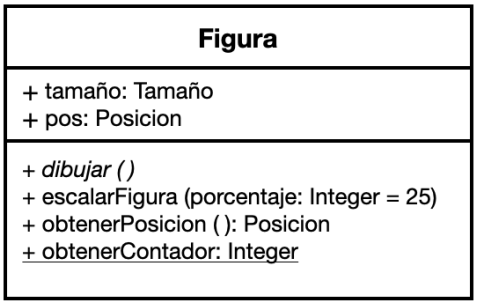


Figura 4–7. Ejemplo de definición de operaciones en una clase

Asociación

Una asociación describe un conjunto de vínculos entre las instancias de las clases. Las asociaciones son las relaciones más generales entre clases, esto es, las relaciones con menor contenido semántico.

Una asociación puede ser:

- Recursiva: conecta una clase consigo misma.
- Binaria: conexión entre dos clases.

- *n*-aria: conecta *n* clases.

Las asociaciones se representan en UML mediante una línea que conecta las clases. Cada conexión de una asociación a una clase se llama extremo de la asociación. En general, las asociaciones son bidireccionales, es decir, no tienen un sentido asociado. En el caso de que se quiera modelar una asociación unidireccional entre clases, deberá indicarse de forma explícita el sentido de la asociación mediante una flecha al final de la línea de asociación. Una flecha indicará que la asociación es navegable en un determinado sentido.

El extremo de una asociación puede ser nombrado, conocido como rol. Un rol representará el subconjunto de instancias de la clase que participa en la asociación. Cada rol tendrá asociada una multiplicidad que indica cuántas instancias pueden estar relacionadas con una única instancia de la clase asociada. La multiplicidad representa un subconjunto abierto de enteros no negativos. La sintaxis para la multiplicidad sería la siguiente:

cota inferior .. cota superior

Donde cota inferior y cota superior son valores de enteros. Adicionalmente, la superior podrá ser un asterisco, que indicará la inexistencia de una cota superior. Un único asterisco indicará un rango de cero a infinito. Un único número indicará una multiplicidad exacta. Por ejemplo, *3* sería equivalente al rango *3..3*. Las multiplicidades más utilizadas son:

- Exactamente una: 1.
- Muchas, de cero a infinito: *.
- Cero o una: 0..1.

La multiplicidad se define con respecto a una instancia de la clase al otro extremo de la asociación.

En la Figura 4–8 puede observarse un ejemplo de una asociación. Esta asociación tiene una navegabilidad bidireccional, pues un componente tiene la responsabilidad de decir a qué montajes pertenece y un montaje tiene la responsabilidad de decir qué componentes lo componen. Por otro lado, respecto a la multiplicidad, un montaje podrá tener 1 o más componentes y un componente estar asociado a 0 o más montajes (para este ejemplo en concreto).

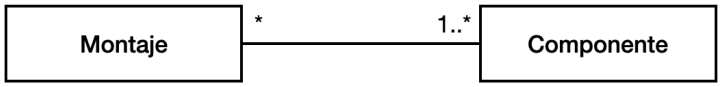


Figura 4–8. Ejemplo de una asociación en diagrama de clases

Agregación y composición

La agregación es una forma de asociación que representa una relación “parte de” entre un agregado y las partes que lo componen.

La existencia de las partes agregadas es independiente de la existencia de la clase agregada, es decir, la clase agregada deberá asociarse con las instancias de las clases de las partes que la forman. Sin embargo, si la instancia de la clase agregada es destruida, las instancias de las clases constituyentes pueden seguir existiendo.

La agregación se representa mediante una asociación en la que el rol del extremo unido a la clase agregada presente un diamante vacío.

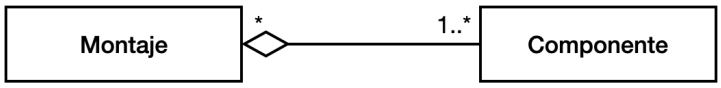


Figura 4–9. Ejemplo de agregación en diagrama de clases

Una variación de la agregación más restrictiva sería la composición, también llamada agregación compuesta. En este caso, las partes agregadas solo pueden ser parte de una clase agregada a la vez y, por ende, la clase agregada es la responsable de la disponibilidad de las partes.

La representación empleada para la composición es la misma que para la agregación con el diamante relleno.

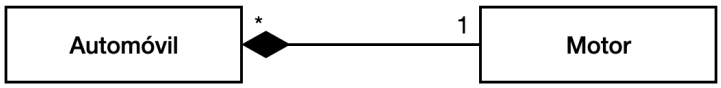


Figura 4–10. Ejemplo de composición en diagrama de clases

Herencia

La herencia es la relación de generalización-especialización entre clases, esto es, una relación entre un elemento general (superclase) y otro más específico (subclase), que está correlacionado al primer elemento y le añade información adicional. Se dice que una superclase es una generalización de las subclases que tiene asociadas y las subclases son especializaciones de una superclase.

La herencia trata de expresar que todo lo que se dice que es cierto para una clase es cierto para sus subclases. La subclase hereda todos los métodos y atributos de la superclase. Por otro lado, se encuentra el principio de capacidad de sustitución, es decir, donde se espere una instancia de la superclase, puede aparecer una instancia de cualquiera de sus subclases.

La herencia se representa mediante una flecha, cuya punta es un triángulo vacío. La flecha va orientada desde la subclase a la superclase.

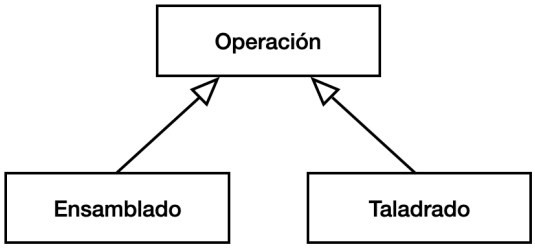


Figura 4–11. Ejemplo de herencia en diagrama de clases

La herencia también puede representarse con una única punta de flecha que llega a la superclase, pero a la base del triángulo que hace de punta de flecha llegan tanto caminos como subclases haya. Esta forma de representación podrá utilizarse para diferenciar distintas relaciones de generalización de una superclase.

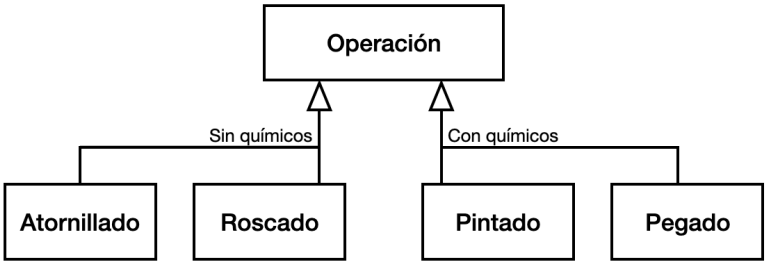


Figura 4–12. Ejemplo de herencia con relaciones en diagrama de clases

4.2.5 Diagrama de secuencias

Los diagramas de secuencias pertenecen a los diagramas de SysML encargados de describir el comportamiento de un sistema, los cuales ya se encontraban en la notación de UML.

Los diagramas de secuencias, al igual que los diagramas de actividades, describen el comportamiento dinámico del sistema. Tanto los diagramas de secuencias como los de actividades informan sobre el orden de las acciones y qué elementos del sistema llevan a cabo las acciones. No obstante, los diagramas de secuencias resultan ser más precisos que los de actividades porque, además, representan el comportamiento en cuanto a la secuencia de mensajes que intercambian los elementos para cumplir con una funcionalidad.

Un diagrama de secuencia está construido a partir de dos dimensiones:

- Horizontal: se encuentran los objetos que participan en la secuencia.
- Vertical: línea de tiempo sobre la que los elementos actúan.

Los diagramas de secuencias están compuestos de dos elementos: objetos y mensajes.

Objeto

Un objeto, también llamado participante, representa a un elemento que formará parte de la interacción. Un objeto resulta ser una instancia y cada uno de los objetos del diagrama representa solamente una instancia de ese objeto y no varias.

Un objeto se representa como un rectángulo del que sale una línea vertical discontinua llamada línea de vida, que representará el tiempo en el que un objeto está presente. En el interior del rectángulo se encontrará el identificador del objeto.

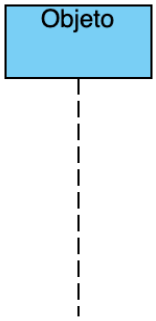


Figura 4–13. Notación de objeto en diagrama de secuencias

Si un objeto existe al principio de la interacción representada en el diagrama, se situará en el eje horizontal del diagrama, por encima del primer mensaje. Así, si el objeto es creado en el intervalo de tiempo representado en el diagrama, la línea comienza en el punto que representa ese instante y encima se coloca el objeto.

Si un objeto es destruido durante la interacción que muestra el diagrama, la línea de vida termina con un aspa, que muestra el final de vida. No obstante, si un objeto no es eliminado, la línea de vida se prolonga hasta el final del diagrama.

Los objetos contienen el denominado foco de control, también llamado casilla de activación, que representa cuándo está activo un objeto. Se representa como un rectángulo delgado superpuesto a la línea de vida del objeto. La parte superior del rectángulo indicará el comienzo de una acción del objeto y la parte inferior su terminación.

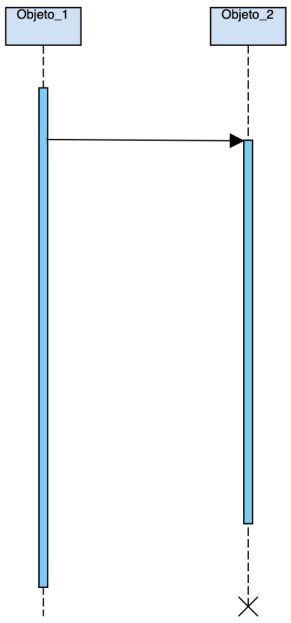


Figura 4–14. Notación de foco de control en diagrama de secuencias

Mensaje

El envío de información de un participante a otro se realiza mediante mensajes. También, un objeto puede

mandarse un mensaje así mismo y no existirá interacción entre dos objetos diferentes, esto podrá representar el procesamiento de cierta información, por ejemplo.

Un mensaje se representa como una flecha continua y horizontal entre las líneas de vida de los objetos que intercambian el mensaje. La flecha va desde el objeto que envía el mensaje al objeto que lo recibe. Si un objeto se manda un mensaje a sí mismo, la flecha comienza y termina en su línea de vida. La flecha incluye el nombre del mensaje y los argumentos que incluye.

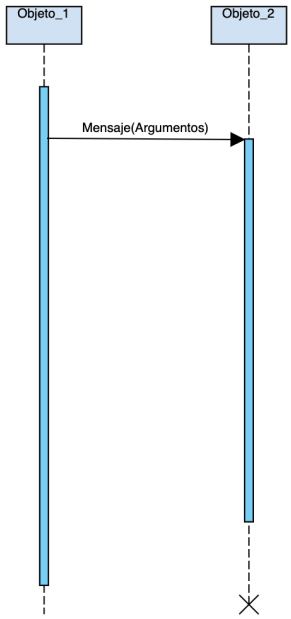


Figura 4–15. Notación de mensaje en diagrama de secuencias

El objeto que recibe el mensaje puede mandar una respuesta. El símbolo de mensaje de respuesta es representado con una flecha discontinua.

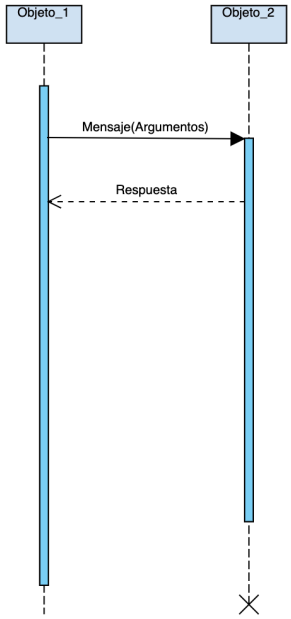


Figura 4–16. Notación de respuesta en diagrama de secuencias

Para crear un objeto se hace uso de un mensaje que tiene como destino el rectángulo que representa el objeto a crear. Esta creación se hace mediante la llamada `<<create>>`. Por otro lado, la destrucción se hará a través de la llamada `<<destroy>>`.

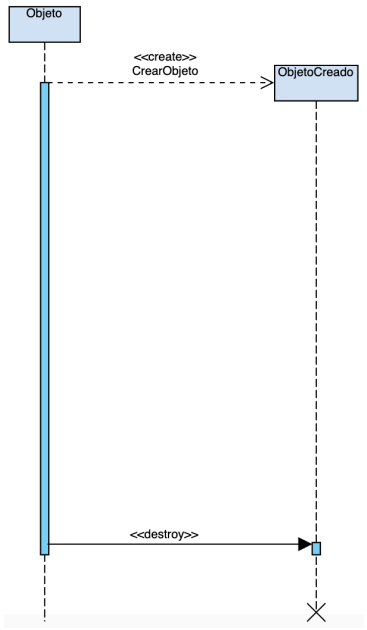


Figura 4–17. Notación para crear y destruir un objeto en diagrama de secuencias

Los diagramas de secuencias también pueden mostrar operaciones iterativas y comportamiento condicional. Estas operaciones son representadas mediante un recuadro de interacción, que encuadra un fragmento del diagrama y en su esquina superior izquierda aparece el operador. Los operadores serían los siguientes:

- `loop`: el fragmento del diagrama se ejecutará múltiples veces.
- `alt`: simboliza una decisión, solo uno de los fragmentos del recuadro se ejecutará dada una condición.

Las expresiones de iteración o condición se dejarán reflejadas entre corchetes.

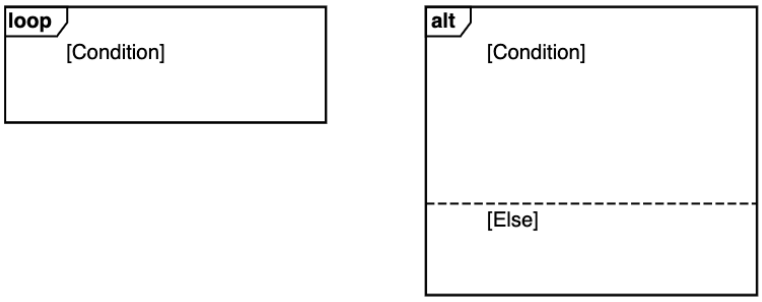


Figura 4–18. Notación de iteración y condición en diagrama de secuencias

4.3 Ontología asociada a los sistemas de simulación

Entre las tecnologías emergentes de la web semántica se encuentra la creación de ontologías para lograr una mejor organización del conocimiento de los modelos. La web semántica propone pasar de una web de documentos a una web de base de datos, donde las máquinas trabajan mejor. Las ontologías proporcionan descripciones exhaustivas y rigurosas, legibles por aplicaciones informáticas. Esta organización del conocimiento ayudará a mejorar la interoperabilidad, la integración y la reutilización.

Las ontologías son utilizadas en el modelado de simulación como herramienta para el intercambio de datos y, de esta forma, poder asegurar la interoperabilidad de la simulación. El uso de las ontologías es una manera eficiente de relacionar las distintas partes intervinientes en un proyecto donde, por ejemplo, un área trabaja con redes de Petri y otra con diagramas de actividades.

En [27] se presenta el diseño y desarrollo de una ontología para el modelado de simulación de eventos

discretos. El desarrollo de dicha ontología está basado en la premisa de que para conseguir una interoperabilidad óptima del sistema se deben considerar también los modelos conceptuales. Así, la ontología no es solo usada para clasificar y organizar conceptos, sino que también codifica las relaciones que existen entre conceptos y captura datos de instancia asociados con conceptos.

Las ontologías asociadas a los sistemas de simulación deben reflejar las técnicas de modelado, los componentes a partir de los cuales se construyen los modelos, así como las reglas que indican cómo deben ejecutarse éstos. Estos tres elementos consistirán en tres clases de nivel superior compuestas de subclases por cada uno de sus formalismos, es decir:

- Superclase Modelo: compuesta de subclases que corresponden a técnicas de modelado existentes en el modelo.
- Superclase Componentes: compuesta de subclases que definen los elementos a partir de los cuales se construye el modelo de simulación.
- Superclase Reglas: compuesta de subclases que indican cómo operan los componentes.

De esta forma, las subclases de cada clase de nivel superior están relacionadas entre sí: una subclase correspondiente a una técnica de modelado en concreto se crea a partir de subclases de Componentes y se ponen en marcha mediante subclases de Reglas. Así, para definir una subclase de Modelo:

1. Se definen las subclases de Componentes adecuadas y se relacionan con la subclase de Modelo.
2. Se definen las subclases de Reglas para explicar cómo trabajan los componentes.
3. Se relacionan las subclases de Reglas con las subclases de Componentes.

Las subclases de Modelo supondrán formalismos de primer nivel que representarán una técnica de modelado en detalle en las que, además de estar conectadas con las subclases de Componentes y Reglas, heredarán subclases inferiores con restricciones y propiedades adicionales. Por ejemplo, estas subclases tratarán la representación del paso del tiempo, el cambio del sistema de un estado a otro, la ubicación dentro de un espacio, qué eventos habilitan un estado o cómo las actividades quedan definidas por unos eventos de inicio y finalización.

4.4 Modelado de simulación basado en MBSE

El enfoque MBSE presenta una serie de actividades relacionadas con el modelado y el análisis de sistemas de fabricación. Si se piensa en el modelo en V, se observa que existe una fuerte relación entre las actividades de desarrollo del proyecto (brazo izquierdo de la V) y las de integración y verificación (brazo derecho de la V), siendo estas últimas de gran importancia en la metodología MBSE. Las actividades de verificación y validación incluyen pruebas y ensayos, donde la simulación puede alcanzar especial relevancia y utilidad.

La metodología MBSE está enfocada al desarrollo y diseño de los sistemas de fabricación, en la que la simulación forma parte de esta metodología, pero no alcanza a su diseño. No obstante, cabe pensar que el modelado de un sistema de simulación, como sistema que es, al igual que uno de fabricación, podría estar sujeto a la metodología MBSE. De esta forma, el desarrollo del sistema de simulación se realizaría paralelamente al modelado del sistema de fabricación.

Actualmente, existe la posibilidad de desarrollar sistemas de simulación de elevada complejidad. Sin embargo, este desarrollo no está respaldado de ninguna metodología en concreto, aunque el enfoque MBSE y sus metodologías pueden ser consideradas la forma adecuada para el desarrollo eficaz y eficiente de los sistemas de simulación. La metodología para el modelado de simulación basado en MBSE contará con las fases típicas de desarrollo de un sistema (definición, diseño, fabricación y control) aplicadas concretamente a un proyecto de simulación.

Uno de los principios fundamentales de la metodología MBSE es el trabajo colaborativo, donde el desarrollo del sistema de fabricación se basa en la intervención de equipos multidisciplinares y participación de todas las partes interesadas. En este sentido, las partes interesadas en el diseño, verificación y validación del sistema de fabricación habrán de estar involucradas también en el desarrollo del sistema de simulación correspondiente. Esta sinergia será fundamental para conseguir la correcta transformación de modelos de fabricación a modelos de simulación ejecutables.

Como se ha visto en apartados anteriores, SysML es el estándar de la metodología MBSE. SysML es usado como lenguaje de diseño. Asimismo, también se ha visto que SysML soporta el diseño y desarrollo de modelos de simulación. Por ello, la utilización del mismo lenguaje en la especificación del sistema de fabricación y su sistema de simulación será relevante a la hora de la transformación de los modelos.

En relación con la transformación de los modelos de fabricación a modelos de simulación ejecutables, SysML es el lenguaje destinado al diseño y será utilizado como base para la elaboración de modelos de simulación ejecutables. En este punto, habrá de intervenir otro lenguaje que permita simular, como puede ser C++. Por ello, teniendo en consideración el enfoque MBSE, habrán de tenerse en cuenta las correspondientes restricciones en las fases de desarrollo conceptual.

En definitiva, el sistema de simulación habrá de ser modelado con el lenguaje de diseño SysML, que podrá servirse de las metodologías que MBSE proporciona para que el sistema resulte ser eficaz y eficiente. Sin embargo, las descripciones del sistema de simulación tendrán que implementarse con otro lenguaje (C++) que permita obtener un modelo de simulación ejecutable. Para ello, será necesario aumentar el nivel de detalle del modelo de simulación añadiendo parámetros y relaciones que permitan la simulación. El modelo ejecutable también tendrá que verificarse y validarse, lo que podrá realizarse mediante una instanciación. La implementación e instanciación del modelo de simulación habrá de servirse de plataformas que soporten la ejecución, como puede ser AnyLogic.

5 SISTEMAS PLM E INTEROPERABILIDAD

El desarrollo de proyectos en base a una metodología MBSE puede traer consigo problemas en relación con la interoperabilidad y colaboración entre modelos. Por ello, es necesario la implementación de un sistema donde queden integrados todos los modelos, un sistema que permita a los modelos operar conjuntamente y sirva como espacio de gestión colaborativa.

5.1 Sistemas PLM

La planificación de recursos empresariales (ERP, por sus siglas en inglés), la gestión del ciclo de vida del producto (PLM, por sus siglas en inglés) y los sistemas de ejecución de fabricación (MES, por sus siglas en inglés) han sido tradicionalmente tres pilares muy distintos del rompecabezas de la tecnología de fabricación. Pero en el mundo actual, donde la entrega rápida y la calidad son sellos distintivos de la empresa es necesario la integración de estos tres elementos que disminuya los tiempos de producción y disminuya los costes de interpretación de datos entre las diferentes áreas.

Al cerrar el círculo entre los sistemas PLM, MES y ERP, los fabricantes esperan facilitar el intercambio de datos entre las áreas funcionales de ingeniería, el taller y la oficina principal. El objetivo es brindar visibilidad que ayude a agilizar los ciclos de entrega de productos, eliminar los procesos manuales redundantes y el desperdicio, y ayudar a identificar y corregir de manera proactiva los problemas de calidad antes de que se vuelvan demasiado costosos y representen barreras para la satisfacción del cliente.

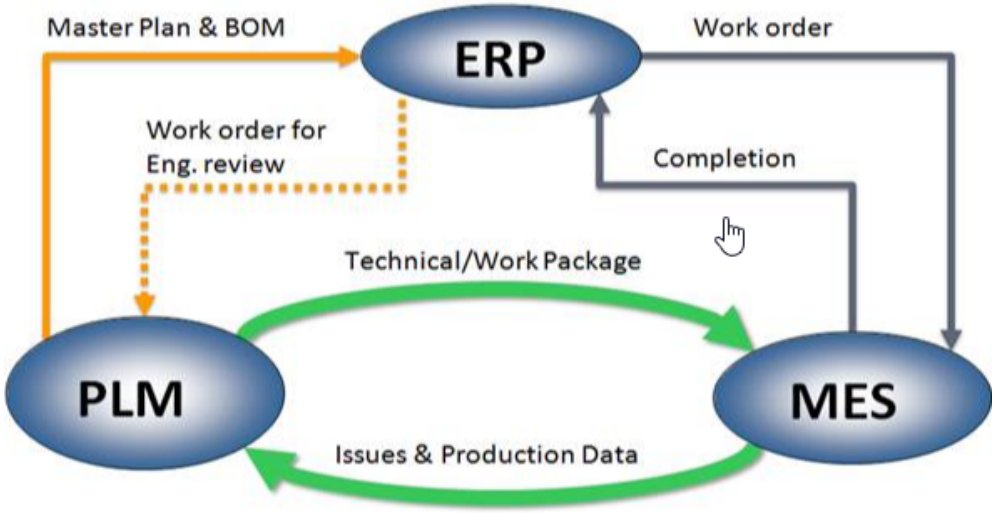


Figura 5–1. Integración sistemas PLM, MES y ERP

La integración de ERP a MES ha sido más estudiada, es una práctica estándar para sincronizar el sistema de registro de los clientes, los datos de pedidos y de inventario con proceso de producción en preparación para cumplir con los requisitos de producción reales y para conciliar el consumo de material para mejor planificación. La integración de PLM a MES, un fenómeno más reciente, se está poniendo de moda a medida que los fabricantes buscan acelerar los tiempos de arranque de los productos y establecer un circuito de retroalimentación entre el diseño y la producción del producto como parte de los esfuerzos continuos de calidad. Crear vínculos entre los tres sistemas sigue siendo una visión a largo plazo para la mayoría, pero los expertos sostienen que será un objetivo necesario para que los fabricantes avancen.

Los sistemas PLM pueden ser definidos desde diferentes puntos de vista. Por ejemplo, desde el punto de vista de la gestión del conocimiento, Ameri y Dutta definen PLM como *"una solución de gestión del conocimiento*

que admite diferentes procesos a lo largo del ciclo de vida del producto dentro de la empresa ampliada" [28]. Por otro lado, se puede encontrar la definición que CIMdata proporciona desde un punto de vista estratégico, pues define PLM como *"un enfoque comercial que aplica un conjunto coherente de soluciones comerciales que respaldan la creación, gestión, difusión y uso colaborativos de la información de productos, apoyando la empresa extendida, abarcando todo el ciclo de vida del producto, e integrando personas, procesos, información y sistemas comerciales"* [29].

PLM representa una visión integral para gestionar todos los datos relacionados con el diseño, la producción, el soporte y la eliminación final de los productos fabricados. Una herramienta PLM se centra exclusivamente en la gestión de datos que cubren la amplitud del ciclo de vida de un producto, sin importar cómo se desarrollen esos datos (CAD, procesadores de texto, hojas de cálculo, informes de problemas, correos electrónicos, etc.). PLM se puede considerar como un depósito de toda la información que afecta a un producto y, a su vez, un proceso de comunicación entre las partes interesadas del producto [30].

Figura 5–2. Sistemas PLM

La complejidad de los productos y la competitividad actual en el mundo empresarial pueden considerarse las razones de mayor relevancia para que las empresas evolucionen hacia una cultura PLM. Los clientes, cada vez más, requieren productos personalizados y en plazos menores, y trabajar con un sistema PLM permite que los cambios en el diseño del producto y su inclusión en el mercado sea más rápido, pues se contará con un reservorio de datos que evitará comenzar el diseño desde cero, acotando así el proceso de desarrollo del producto. Por otro lado, trabajar en espacios colaborativos puede incrementar la innovación, favoreciendo consecuentemente la competitividad. Estos espacios colaborativos facilitan también la inclusión de terceros en la cadena de suministro.

Los sistemas PLM servirán como soporte para implementar los modelos de fabricación, es decir, el sistema PPR. En este sistema se encontrarán todos los datos necesarios para la gestión completa del ciclo de vida de un proyecto.

El modelo integrado resultante del modelado de fabricación y el modelado de simulación puede quedar implementado en un sistema PLM, que proporcionará todos los datos necesarios para la ejecución de la simulación a través de un software de simulación.

5.2 Interoperabilidad

Cuando en el diseño de un sistema existen distintas partes interesadas, puede que los modelos no acaben siendo desarrollados de forma homogénea, lo que supondría que los modelos no pudieran operar conjuntamente. Por ello, han de diseñarse técnicas que permitan la interoperabilidad entre modelos.

La colaboración e interacción entre los elementos que forman un sistema resulta fundamental para evitar la pérdida de información y los problemas de calidad que esto puede conllevar, con sus respectivos costes e inversiones en tiempos innecesarios y evitables. Por ello, compartir datos, información y conocimiento, así como la utilización de la información compartida son los hitos por perseguir para que los sistemas sean interoperables.

La interoperabilidad hace uso de las ontologías con el propósito de garantizar un vocabulario común y compartido. Las ontologías tienen una semántica asociada que permite mapear los datos a nivel conceptual y de forma estándar, consiguiendo que puedan ser utilizados por diferentes aplicaciones. Las ontologías pueden definir conceptos, capturar conocimiento y establecer relaciones de diseño y fabricación de manera formal y no ambigua.

Los modelos ontológicos de un sistema contarán con gran cantidad de datos que habrán de servirse de sistemas PLM que almacenen los modelos durante todo el ciclo de vida de un proyecto, permitiéndose de esta forma la reutilización de modelos y el trabajo en un espacio colaborativo y adecuado para cada una de las partes interesadas en el proyecto. Los sistemas PLM proporcionarán la interoperabilidad necesaria entre los modelos ontológicos.

Las ontologías representan el conocimiento en un lenguaje que puede ser utilizado por los softwares. Por ello, el modelo integrado resultante del modelado de fabricación y el modelado de simulación puede quedar definido en base a un entorno ontológico, de manera que el modelo sea genérico e interoperable. Este modelo integrado quedará implementado en un sistema PLM, que proporcionará todos los datos para su simulación. Para ello, se realizará una instanciación, donde los elementos del modelo ontológico tomarán un valor concreto y representativo del mundo real.

6 SOFTWARE DE SIMULACIÓN

Un proceso productivo puede ser considerado un sistema de eventos discretos, es decir, existen unos eventos que ocurren en instantes espaciados en el tiempo que cambian el estado del sistema. Por ejemplo, el estado de una máquina se altera ante la ocurrencia de alguno de estos dos eventos: (1) activación al ingreso de una pieza a procesar, o (2) parada tras la finalización del procesado de una pieza.

Existe una gran variedad de softwares de simulación por ordenador. En concreto, los programas de simulación de eventos discretos empezaron a aparecer en la década de los 60. AnyLogic será el software de simulación que se usará en este trabajo.

6.1 Introducción a AnyLogic

AnyLogic es un software de simulación creado por la empresa estadounidense The AnyLogic Company, lanzado al mercado en el año 2000.

AnyLogic será usado en este proyecto para modelar y simular los diferentes procesos productivos basados en eventos discretos. No obstante, este software permite también crear y combinar ambientes de simulación de sistemas dinámicos y simulación basada en agentes.

AnyLogic denomina a los modelos de simulación *Projects*. Los *Projects*, los sistemas reales modelados, son creados en términos de agentes, procesos y recursos. Estos modelos se desarrollan gráficamente en forma de diagramas de flujo.

Los agentes pueden representar diferentes entidades: clientes, pacientes, documentos, productos, vehículos, proyectos, ideas, compañías, países, etc. Los agentes son los módulos de construcción principales del modelo de AnyLogic, se tratan de la unidad de diseño del modelo.

Los recursos son aquellas unidades que los agentes pueden tomar y liberar. Como ejemplos, se podrían citar los siguientes: personal, operarios, ordenadores, transportes, equipos, etc.

En el espacio de trabajo de los modelos en AnyLogic se pueden distinguir cuatro áreas principales (véase Figura 6–1):

- *Project Window*: esta ventana permite el acceso a los distintos proyectos y a sus elementos.
- *Palette Window*: proporciona la lista de elementos para desarrollar el diagrama. Esta ventana comparte espacio con la *Project Window*.
- *Diagram Editor Window*: espacio donde se desarrolla el diagrama que define la estructura de un modelo, aquí quedan definidos eventos, comportamientos, gráficos…
- *Properties Window*: en este espacio se permite ver y modificar las propiedades de los componentes de un proyecto.

La creación en AnyLogic de modelos de eventos discretos se realizará mediante la *AnyLogic Process Modeling Libary*. Los objetos de esta librería permitirán modelar el mundo real en términos de agentes, procesos y recursos.

Los diagramas de flujo se crearán agregando módulos, los objetos, desde la *Palette Window* al *Diagram Editor Window*, conectándose entre sí y ajustando sus parámetros mediante la *Properties Window*. Los diagramas de flujo en AnyLogic son jerárquicos, escalables, extensibles y orientados a objetos, lo que permite al usuario modelar sistemas complejos a cualquier nivel de detalle.

Los módulos a los que se hacía referencia anteriormente son denominados en AnyLogic como *blocks*. Generalmente, el modelo comienza con un *Source block* que genera los agentes, las entidades, y los introduce en el proceso. El modelo termina con un *Sink block* que elimina los agentes del proceso.

Los modelos no suelen ser creados de la forma que anteriormente se describía, la cual consiste básicamente en crear un gráfico arrastrando y soltando objetos. Este modo de crear los gráficos no es eficaz cuando existe la necesidad de usar distribuciones de probabilidad, evaluar expresiones y condiciones de prueba que contienen propiedades de diferentes objetos, definir datos personalizados o diseñar algoritmos. Por ello, en la práctica, estas tareas se realizan de mejor forma en formato texto. Cualquier herramienta de modelado de simulación incluye un código en formato texto.

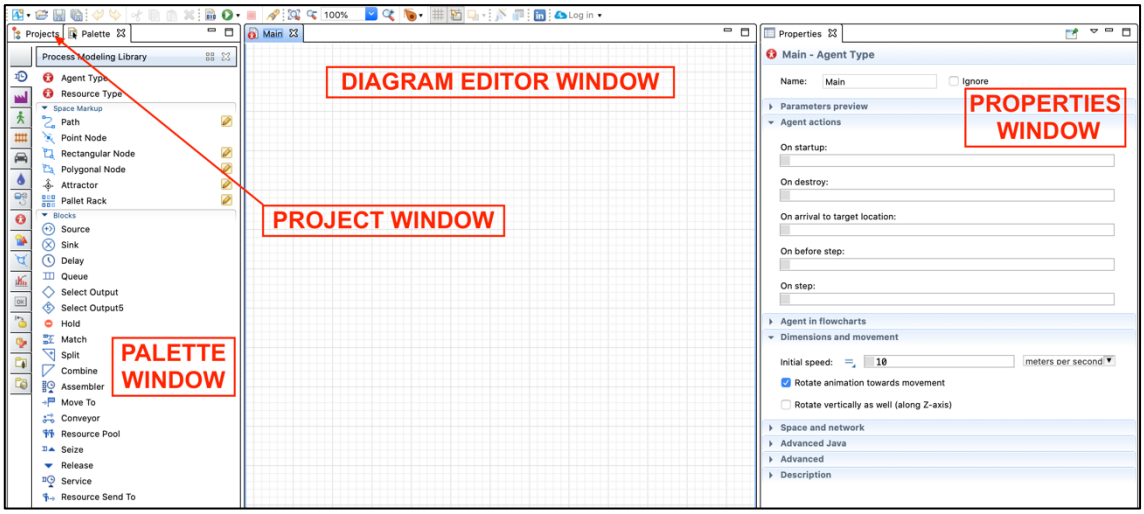


Figura 6–1. Espacio de trabajo en AnyLogic

Un modelo desarrollado en AnyLogic está completamente descrito en código Java, que es un lenguaje de programación orientado a objetos. Estos códigos son archivos XML con datos de texto.

Cualquier modelo insertado en el gráfico, sus condiciones con otros módulos, funciones o propiedades definidas tendrán detrás un código Java. Además, de forma recíproca, cualquier edición en el código Java se verá modificado en el diagrama de flujo. Así, los modelos podrán crearse a partir de archivos en XML y se cargarán en el software AnyLogic para ejecutar el modelo y observar la simulación. A continuación, se estudiará cómo modelar en AnyLogic y qué código se genera en el fichero XML con cada objeto, así como las instancias que se van desarrollando.

6.2 Modelos de simulación mediante fichero XML

XML es un conjunto de reglas que son legibles tanto por humanos como por máquinas. Por eso, este código es útil cuando los datos deben ser creados o verificados por humanos, pero interpretados por máquinas.

Básicamente, XML consiste en marcado y contenido. La construcción más común es encerrar el contenido por una etiqueta de inicio y final, como marcado:

```
<Etiqueta>Contenido</Etiqueta>
```

Figura 6–2. Marcado en XML, primera forma

Es lo mismo que decir:

```
<Etiqueta>
    Contenido
</Etiqueta>
```

Figura 6–3. Marcado en XML, segunda forma

La etiqueta de inicio, el contenido y la etiqueta final forman un elemento. El contenido puede contener uno o

varios elementos, que luego se denominan elementos secundarios. Se crea así una estructura jerárquica. Por ejemplo, a modo muy reducido, un *Source block* en el diagrama de flujo correspondería a:

```
<EmbeddedObject>
    <Name><![CDATA[EntradaPiezas]]></Name>
    <Parameter>
        <Name><![CDATA[maxArrivals]]></Name>
        <Value Class="CodeValue">
            <Code><![CDATA[100]]></Code>
        </Value>
    </Parameter>
</EmbeddedObject>
```

Figura 6–4. Estructura jerárquica en XML

Estas estructuras se verán con mayor detalle en los próximos apartados y se conocerá qué genera cada módulo del diagrama de flujo en el fichero XML.

Como anotación, la secuencia `<![CDATA[…]]>` marca para que se interprete como una cadena de caracteres y no como contenido etiquetado.

El fichero de texto del modelo, de forma genérica, contiene las siguientes etiquetas a destacar:

- Etiqueta `ActiveObjectClass`: esta etiqueta se encuentra dentro de una etiqueta genérica llamada `ActiveObjectClasses`. En esta etiqueta se encontrará toda la información relativa a los agentes.
- Etiqueta `EmbeddedObject`: esta etiqueta se encuentra en una etiqueta genérica llamada `EmbeddedObjects`, a su vez dentro de `ActiveObjectClasses`. Aquí se incluirá toda la información relativa a los objetos del modelo.
- Etiqueta `Experiments`: define los datos a simular.

En la siguiente figura se puede observar la estructura general del fichero XML de un modelo en AnyLogic. Se trata de un modelo que no contiene aún ningún objeto, pero se pueden distinguir, como ejemplo, las etiquetas `Name` y `ModelTimeUnit`, estos son los primeros datos que se definen al crear un nuevo modelo en AnyLogic.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!--
3   ***************************************************
4                AnyLogic Project File
5   ***************************************************
6   -->
7   <AnyLogicWorkspace WorkspaceVersion="1.9" AnyLogicVersion="8.5.2.202001241834" AlpVersion="8.5.1">
8   <Model>
9       <Id>1594230068064</Id>
10      <Name><![CDATA[Ejemplo]]></Name>
11      <EngineVersion>6</EngineVersion>
12      <JavaPackageName><![CDATA[ejemplo]]></JavaPackageName>
13      <ModelTimeUnit><![CDATA[Minute]]></ModelTimeUnit>
14      <ActiveObjectClasses>
15          <!--   =========   Active Object Class   ========  -->
16          <ActiveObjectClass...>
153     </ActiveObjectClasses>
154     <DifferentialEquationsMethod>EULER</DifferentialEquationsMethod>
155     <MixedEquationsMethod>RK45_NEWTON</MixedEquationsMethod>
156     <AlgebraicEquationsMethod>MODIFIED_NEWTON</AlgebraicEquationsMethod>
157     <AbsoluteAccuracy>1.0E-5</AbsoluteAccuracy>
158     <FixedTimeStep>0.001</FixedTimeStep>
159     <RelativeAccuracy>1.0E-5</RelativeAccuracy>
160     <TimeAccuracy>1.0E-5</TimeAccuracy>
161     <Frame...>
165     <Database...>
174
175     <RunConfiguration...>
199     <Experiments...>
257 </Model>
258 </AnyLogicWorkspace>
```

Figura 6–5. Estructura general del fichero XML

En los siguientes apartados se irán desarrollando en detalle distintos módulos de AnyLogic necesarios para la creación de un proceso productivo industrial. A su vez, se irán conociendo los parámetros y relaciones correspondientes más importantes entre el fichero XML del modelo y el diagrama de flujo.

6.3 Modelado mediante AnyLogic

6.3.1 Agente y atributos

Los agentes son la unidad de diseño de un modelo. Esta unidad de diseño puede tener un comportamiento determinado, memoria, tiempo…

El diseño de un agente comienza, generalmente, con la identificación de sus atributos, comportamiento y relación con el mundo externo.

Los tipos de agentes son desarrollados por el usuario. Para crear un nuevo agente se debe insertar el elemento *Agent* desde la *Agent Palette*. Se podrá crear una población de agentes, un solo agente o definir el tipo de agente y no crear ningún agente de este tipo todavía.

El agente puede tener atributos, AnyLogic los llama parámetros. Los atributos se usan para representar características del objeto modelado. Hay una clara diferencia entre variables y parámetros. Una variable representa un estado del modelo y puede cambiar durante la simulación, se tratan de variables locales. Por el contrario, un parámetro se usa para describir objetos estáticamente, son variables globales. Un parámetro es una constante en una sola simulación y se cambia solo cuando necesita ajustar el comportamiento de su modelo.

Para crear un parámetro se insertará el elemento *Parameter* desde la *Agent Palette*. Habrá de definirse el tipo de parámetro correspondiente.

Como ejemplo: existe un sistema donde llegan dos tipos de piezas, cada tipo de pieza tiene una ruta a seguir y ambas piezas concurren en una estación común, por ejemplo, una estación de torneado, donde en función del tipo de pieza, el procesado tendrá un tiempo u otro. En ese caso, al sistema entrarán agentes de tipo pieza con dos atributos: tipo de pieza y tiempo de torno. Habrá de crearse en el modelo el agente `Pieza` y definirle los dos parámetros `TipoPieza` (tipo entero) y `TiempoTorno` (tipo real):

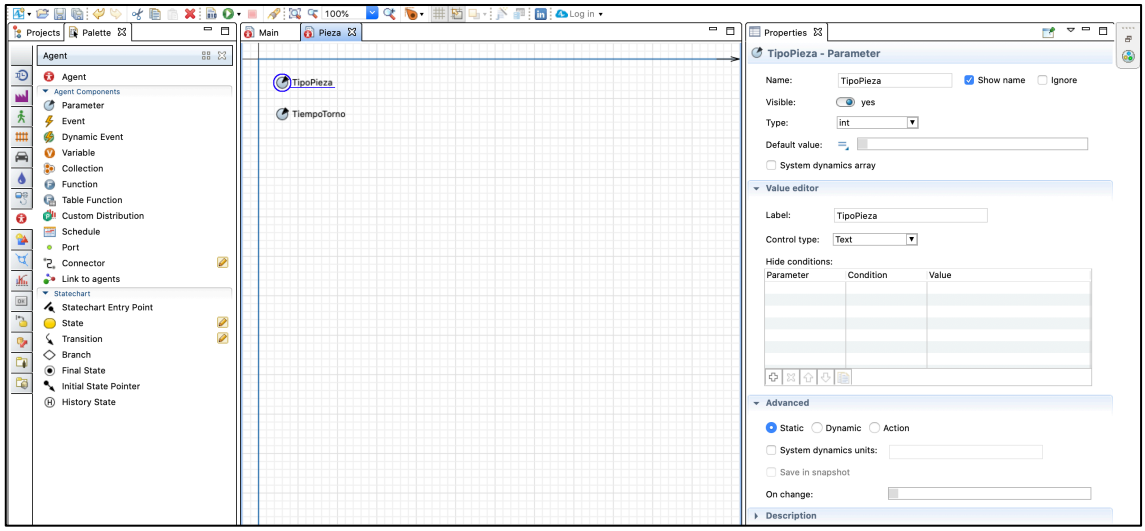


Figura 6–6. Agente y sus atributos en el diagrama

Como se definió anteriormente, el agente *Pieza* será definido en el fichero XML en una etiqueta `ActiveObjectClass`.

Por defecto, al crear un modelo en AnyLogic, en la etiqueta `ActiveObjectClasses` se encontrará una etiqueta secundaria `ActiveObjectClass` para el `Main` del modelo, que se trata como un agente. En el agente `Main` se irá desarrollando el diagrama que define la estructura del modelo. Ahora, al haberse creado un

nuevo agente, se encontrará una nueva etiqueta secundaria `ActiveObjectClass` para el agente `Pieza`.

Figura 6–7. Etiquetas ActiveObjectClass

Se encontrará el siguiente código dentro de la etiqueta `ActiveObjectClass` para el agente `Pieza`. Se observa cómo se definen los dos atributos desarrollados, parámetros para AnyLogic.

```
<ActiveObjectClass>
    [...]
    <Name><![CDATA[Pieza]]></Name>
    [...]
    <GenericParameter>
        <Id>1594317413987</Id>
        <Name><![CDATA[1594317413987]]></Name>
        <GenericParameterValue Class="CodeValue">
            <Code><![CDATA[T extends Agent]]></Code>
        </GenericParameterValue>
        <GenericParameterLabel><![CDATA[Generic parameter:]]></GenericParameterLabel>
    </GenericParameter>
    [...]
    <Variables>
        <Variable Class="Parameter">
            [...]
            <Name><![CDATA[TipoPieza]]></Name>
            [...]
            <Properties SaveInSnapshot="true" ModificatorType="STATIC">
                <Type><![CDATA[int]]></Type>
                [...]
            </Properties>
        </Variable>
        <Variable Class="Parameter">
            [...]
            <Name><![CDATA[TiempoTorno]]></Name>
            [...]
            <Properties SaveInSnapshot="true" ModificatorType="STATIC">
                <Type><![CDATA[double]]></Type>
                [...]
            </Properties>
        </Variable>
    </Variables>
    [...]
</ActiveObjectClass>
```

Figura 6–8. XML de un agente y sus atributos

6.3.2 Entrada de agentes al modelo

Un modelo comienza generalmente por módulo *Source*, que genera los agentes. Los agentes generados pueden ser de tipo estándar o de cualquier otro tipo creado por el usuario, esto definirá en el campo *New agent* de las propiedades de este módulo. Otra de las propiedades interesante que tiene este módulo es que permite definir cuándo y cuántos agentes se van a generar. El campo *Arrivals defined by* permite definir cuándo se van a generar los agentes. Entre las opciones existentes, se pueden destacar las siguientes:

- *Rate*: los agentes se generan a la velocidad de llegada especificada, la frecuencia con la que llegar los agentes al sistema. La tasa de llegada se indicará en el campo *Arrival rate*.
- *Interravial time*: el tiempo entre dos agentes se define mediante la expresión especificada. La expresión especificada se indica en el campo *Interravial time*, expresión que suele evaluar el tiempo de retraso de cada agente. Además, habrá de definirse cuándo se producirá la primera entrada en el campo *First arrival occurs* y podrá ser: después del tiempo de espera, al inicio del modelo o en un tiempo especificado.

Por otro lado, se pueden delimitar cuántos agentes se van a generar. Por ello, se marcará el campo *Limited number of arrivals* y se indicará la cantidad en el campo *Maximum number of arrivals*.

Como ejemplo: se supone que a un sistema llegan agentes del tipo `Pieza`, como los creados en el apartado anterior, y que las piezas llegan con una tasa de 0.3 piezas por minuto y llegarán un máximo de 100 piezas. Estas propiedades se definirían de la siguiente forma:

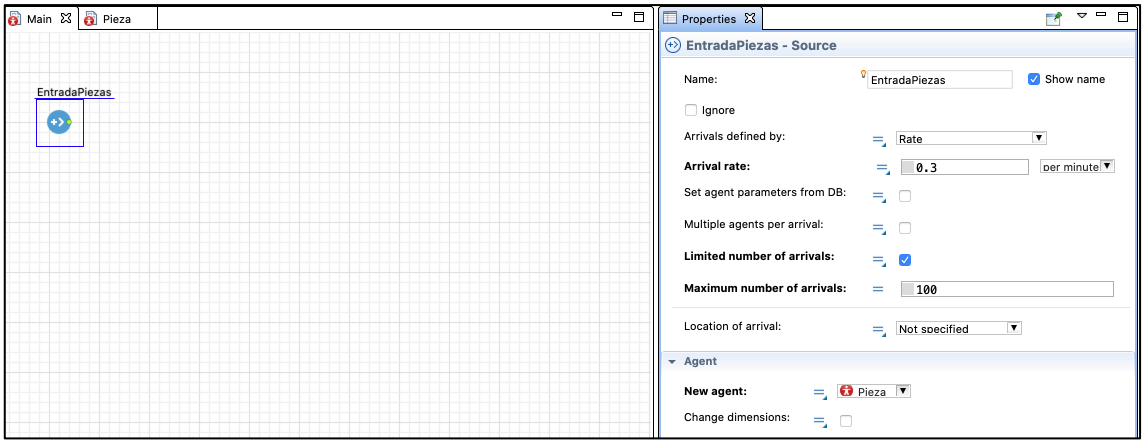


Figura 6–9. Generar agentes con intervalo y máximo de llegadas

Los módulos insertados en el modelo se encontrarán en su etiqueta `EmbeddedObject` correspondiente del fichero XML. Estas etiquetas son creadas dentro de la etiqueta `ActiveObjectClass` del agente `Main` y una etiqueta genérica `EmbeddedObjects`.

En la siguiente figura se muestra la correspondencia del módulo *Source*, creado anteriormente en el software de simulación, en el fichero XML. Se puede destacar lo siguiente:

- Etiqueta marcando el nombre del módulo.
- Etiqueta `ActiveObjectClass`, que activa un nuevo objeto.
- Etiqueta `GenericParameterSubstitute`, que especifica el nuevo objeto.
- Etiquetas de parámetros con la definición de las propiedades, según modelo en AnyLogic: *Arrivals defined by* y su valor, *Maximum numbers of arrivals*.
- Agentes que se crean según campo *New Agent* (véase la equivalencia entre agente y entidad).

```
<EmbeddedObject>
    <Id>1594744835428</Id>
    <Name><![CDATA[EntradaPiezas]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Source]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Source]]></ClassName>
            <ItemName><![CDATA[1412336242928]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[arrivalType]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[rate]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[0.3]]></Code>
                <Unit Class="RateUnits"><![CDATA[PER_MINUTE]]></Unit>
            </Value>
        </Parameter>
        [...]
        <Parameter>
            <Name><![CDATA[limitArrivals]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maxArrivals]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[100]]></Code>
            </Value>
        </Parameter>
       [...]
        <Parameter>
            <Name><![CDATA[newEntity]]></Name>
            <Value Class="EntityCodeValue">
                <EntityEmbeddedObject>
                    <ActiveObjectClass>
                        <PackageName><![CDATA[ejemplo]]></PackageName>
                        <ClassName><![CDATA[Pieza]]></ClassName>
                    </ActiveObjectClass>
                    <GenericParameterSubstitute>
                        <GenericParameterSubstituteReference>
                            <PackageName><![CDATA[ejemplo]]></PackageName>
                            <ClassName><![CDATA[Pieza]]></ClassName>
                            <ItemName><![CDATA[1594317413987]]></ItemName>
                        </GenericParameterSubstituteReference>
                    </GenericParameterSubstitute>
                    <Parameters>
                        <Parameter>
                            <Name><![CDATA[TipoPieza]]></Name>
                        </Parameter>
                        <Parameter>
                            <Name><![CDATA[TiempoTorno]]></Name>
                        </Parameter>
                    </Parameters>
                    [..]
                </EntityEmbeddedObject>
            </Value>
        </Parameter>
        [...]
    </Parameters>
    <ReplicationFlag>false</ReplicationFlag>
    [...]
</EmbeddedObject>
```

Figura 6–10. Entrada agentes con tasa y máximo de llegadas en XML

Si ahora se supusiera que el tiempo de espera entre dos entradas de agentes es 0.5 minutos, que la primera llegada se produce al iniciar el modelo y que hay un máximo de 100 llegadas, el modelo sería:

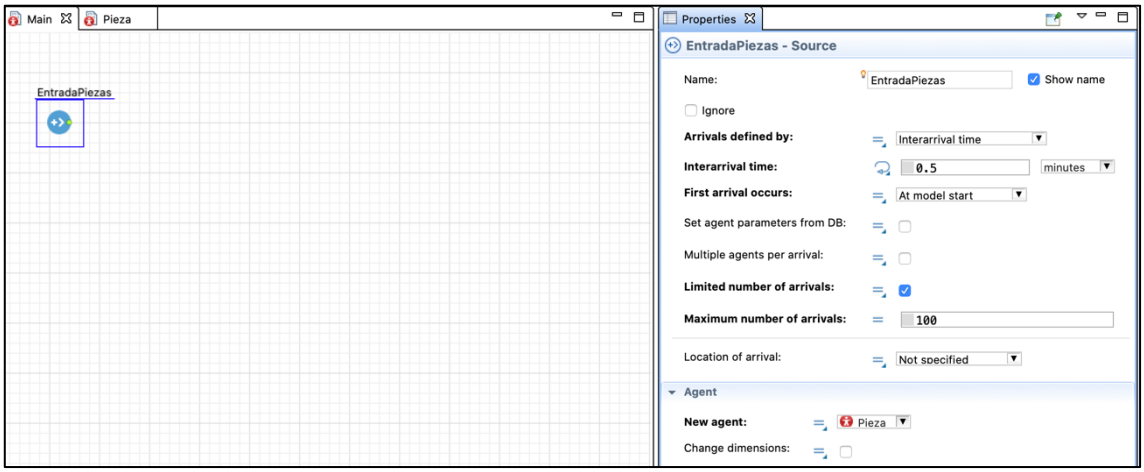


Figura 6–11. Generar agentes con intervalo y máximo de llegadas

El fichero XML quedaría modificado de la siguiente forma (se muestran solo las diferencias respecto al otro fichero):

```
<EmbeddedObject>
    <Id>1594744977565</Id>
    <Name><![CDATA[EntradaPiezas]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Source]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Source]]></ClassName>
            <ItemName><![CDATA[1412336242928]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[arrivalType]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.INTERARRIVAL_TIME]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[rate]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[interarrivalTime]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[0.5]]></Code>
                <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[firstArrivalMode]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.AT_START]]></Code>
            </Value>
        </Parameter>
        [...]
        <Parameter>
            <Name><![CDATA[limitArrivals]]></Name>
            <Value Class="CodeValue">
```

Figura 6–12. Diferencias en XML entre entradas definidas por una tasa o por tiempo de espera

Un módulo *Source* tiene, por defecto, las llegadas de agentes definidas por una tasa (*Rate*). Por ello, en la Figura 6–10 se puede observar que la etiqueta `arrivalType` no tiene ningún valor asignado, al contrario que en la Figura 6–12. No obstante, la opción de *Rate* se puede dejar reflejado según la figura siguiente:

```
<Parameter>
    <Name><![CDATA[arrivalType]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[self.RATE]]></Code>
    </Value>
</Parameter>
```

Figura 6–13. Etiqueta arrivalType para llegadas de agentes según Rate

De igual forma ocurre cuando entre la llegada de un agente y otro existe un tiempo de espera (*Interarrival time*) y la primera entrada ocurre después del tiempo de espera (*Aflter timeout*), que es su valor por defecto. En ese caso, la etiqueta `firstArrivalMode` no tiene ningún valor asignado. También esto se puede dejar reflejado, según la figura siguiente:

```
<Parameter>
    <Name><![CDATA[firstArrivalMode]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[self.AFTER_TIMEOUT]]></Code>
    </Value>
</Parameter>
```

Figura 6–14. Etiqueta firstArrivalMode para After timeout

6.3.3 Funciones

AnyLogic permite definir sus propias funciones. Una función devolverá el valor de una expresión cada vez que el usuario la llame desde el modelo. Las funciones son escritas en Java.

Las funciones se modelan insertando el elemento *Function* desde la *Agent Palette* y, posteriormente, se establece su código en su vista de propiedades.

Una función puede no devolver nada y solo realizar algunas acciones. Para ello, en las propiedades del elemento *Function* se marcará el campo *Just action (returns nothing)*. Por el contrario, la función puede devolver un resulta, que tendrá un tipo concreto (entero, booleano…). En ese caso, habrá de marcarse el campo *Returns value* y definir el tipo en el campo *Type*.

La función tendrá unos argumentos, que se definirán en la sección *Arguments* de las propiedades del elemento. Los argumentos son definidos en una lista que tiene dos columnas: nombre y tipo del argumento.

Finalmente, se escribirá el cuerpo de la función, también en la vista de propiedades del elemento.

Siguiendo el ejemplo que se empezó en el apartado 6.3.1 y al sistema llegaran los dos tipos de piezas con la siguiente probabilidad: el 40 % son de tipo 1 y el 60 % son de tipo 2. Así, cada vez que se genere una pieza (un agente) ésta será con un valor determinado: tipo 1 o tipo 2. Para conseguir implementar la cadencia de cada tipo, que anteriormente se indicaba, será necesario implementar una función que determine ese valor (el tipo) y almacene en un parámetro (`TipoPieza`).

Se definirá la función `ObtenerTipoPieza`, que devolverá un valor entero: 1 ó 2, en función del tipo. La función tendrá como argumentos `listaTipos` y `listaProbabilidad`. Se define `listaTipos` como un vector de enteros en el que se reflejan los tipos de piezas que se tiene: {1, 2}. Se define `listaProbabilidad` como un vector de flotantes en el que se reflejan las probabilidades de cada uno de los tipos de piezas existentes: {0.4, 1.0}. El cuerpo de la función será el siguiente:

```
double numeroAleatorio=uniform();
    int i=0;
    while (listaProbabilidad[i]<numeroAleatorio)
          i++;
return listaTipos[i];
```

En esta función se usa una distribución uniforme entre 0 y 1, esta es una función existente en la librería de AnyLogic que generará una muestra en el intervalo [0, 1). Sabiendo que la probabilidad se distribuye uniformemente a lo largo de dicho intervalo, la variable `numeroAleatorio` alcanzará en un 40 % de las

veces un valor menor que `listaProbabilidad[0]`, es decir, 0.4. En ese caso, observando el cuerpo de la función, ésta devolverá `listaTipos[0]` y el tipo de pieza que se generará será de tipo 1. Por el contrario, la variable `numeroAleatorio` alcanzará en un 60 % de las veces un valor mayor o igual que `listaProbabilidad[0]`. Ahora, en ese caso, la función devolverá `listaTipos[1]` y el tipo de pieza que se generará será de tipo 2. Se definió `listaProbabilidad[1]` como 1.0, y no 0.6, para obligar la salida del bucle declarado.

Habiendo definido la función `ObtenerTipoPieza`, el módulo `EntradaPiezas` habrá de llamar a esa función para que la pieza se genere con su atributo `TipoPieza` correspondiente. Esta llamada se registrará en las acciones de las propiedades del módulo `EntradaPiezas` para que la pieza salga con su atributo definido, esto se indica en el campo *On at exits* de la vista de acciones de las propiedades del módulo. La llamada a la función sería la siguiente:

```
agent.TipoPieza=ObtenerTipoPieza(new int[] {1,2}, new double[]
{0.4,1.0});
```

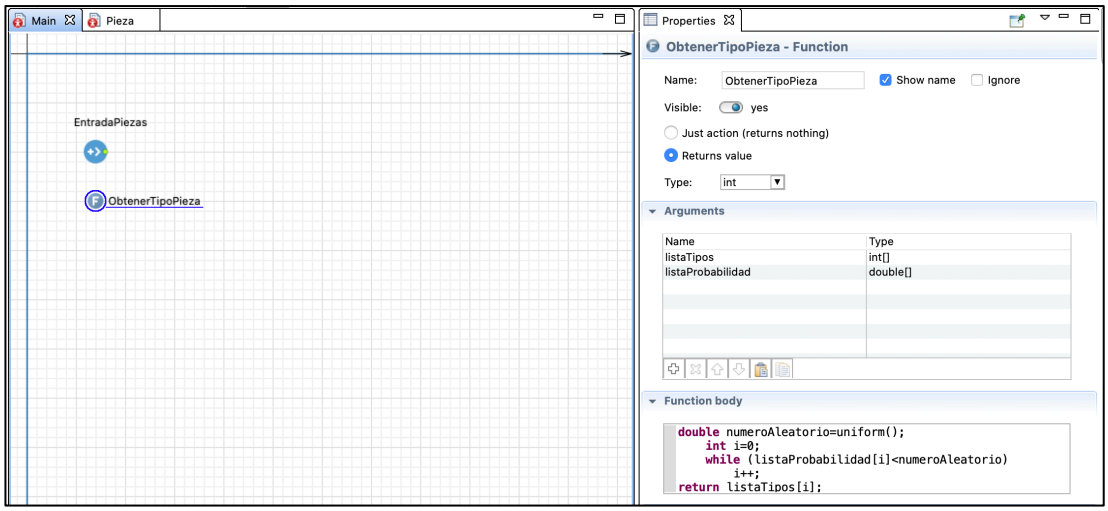


Figura 6–15. Modelado de una función

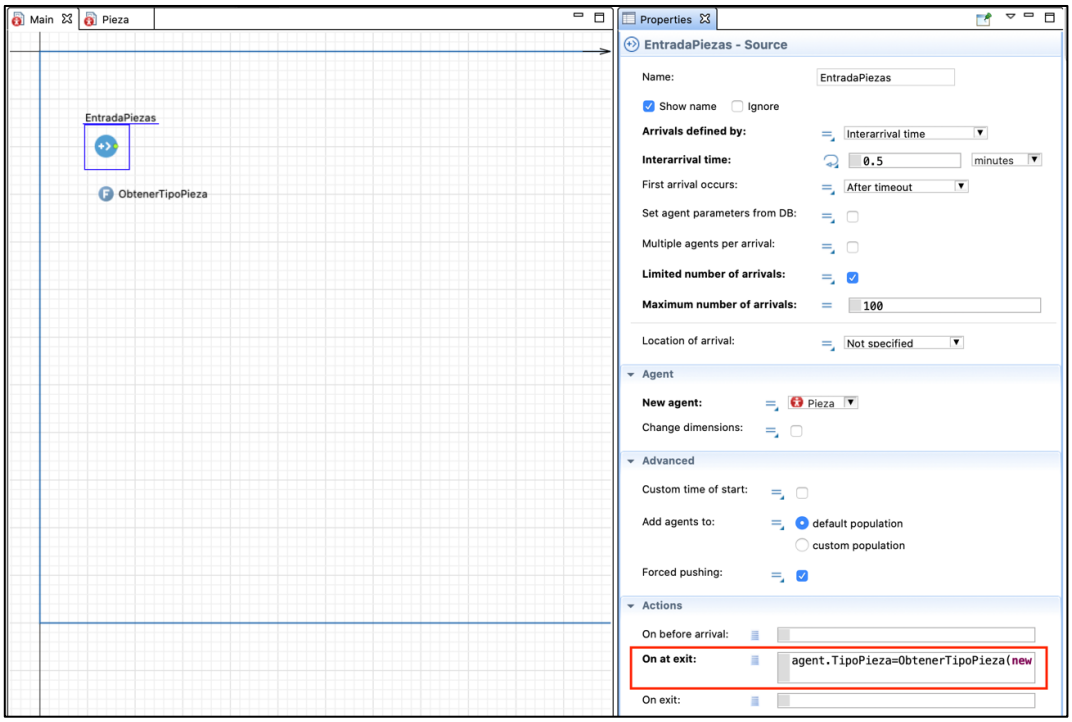


Figura 6–16. Llamada a una función

En el fichero XML, al crear una función en el modelo de AnyLogic, se establece una nueva etiqueta para las funciones dentro de la etiqueta `ActiveObjectClass` del agente `Main`: etiqueta `Functions`, que abarcará todas las funciones definidas en el modelo. Dentro de esta etiqueta genérica se encontrará la etiqueta `Function` para la función anteriormente diseñada.

En el código XML se puede observar lo siguiente:

- Etiqueta `ReturnModificator`: indicará si la función devuelve un resultado o no.
- Etiqueta `ReturnType`: indicará el tipo de resultado, si es el caso que la función devuelve un resultado.
- Marcado del nombre de la función.
- Etiquetas `Parameter`: definen los argumentos de la función. Estas etiquetas incluyen el nombre del argumento y su tipo.
- Etiqueta `Body`: incluye el cuerpo de la función.

```
<Functions>
    <Function AccessType="default" StaticFunction="false">
        <ReturnModificator>RETURNS_VALUE</ReturnModificator>
        <ReturnType><![CDATA[int]]></ReturnType>
        <Id>1594917157753</Id>
        <Name><![CDATA[ObtenerTipoPieza]]></Name>
        [...]
        <Parameter>
            <Name><![CDATA[listaTipos]]></Name>
            <Type><![CDATA[int[]]]></Type>
        </Parameter>
        <Parameter>
            <Name><![CDATA[listaProbabilidad]]></Name>
            <Type><![CDATA[double[]]]></Type>
        </Parameter>
        <Body><![CDATA[double numeroAleatorio=uniform();
            int i=0;
            while (listaProbabilidad[i]<numeroAleatorio)
                i++;
            return listaTipos[i];]]></Body>
    </Function>
</Functions>
```

Figura 6–17. XML de una función

Por otro lado, al haberse añadido una nueva propiedad en el módulo `EntradaPiezas`, también habrá cambios en el código correspondiente a este módulo. Dentro de la etiqueta `EmbeddedObject` correspondiente al módulo `EntradaPiezas`, como una propiedad más, aparecerá dentro de la etiqueta secundaria `Parameters` un nuevo parámetro con la nueva propiedad. Esta nueva propiedad se trata de la llamada a la función `ObtenerTipoPieza` para dar valor al atributo `TipoPieza` del agente que salga del módulo `EntradaPiezas`.

```
<Parameter>
    <Name><![CDATA[onAtExit]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[agent.TipoPieza=ObtenerTipoPieza(new int[] {1,2},new double[] {0.4,1.0});]]></Code>
    </Value>
</Parameter>
```

Figura 6–18. Llamada a una función a la salida de un módulo Source

6.3.4 Dos puertos de salida

En el caso que se tenga que dividir en dos el flujo de una cadena productiva, habrá de usarse el módulo *SelectOutput*. Este módulo permite clasificar a los agentes de acuerdo con ciertos criterios. Este módulo tiene una entrada y dos puertos de salida, los agentes saldrán por una salida u otra en función de la condición impuesta, que puede depender del agente o de cualquier factor externo. Los agentes no pasan tiempo en este módulo.

El módulo *SelectOutput* tiene una salida verdadera y otra falsa, es decir, los agentes que cumplan la condición

impuesta saldrán por la salida verdadera (salida de la derecha) y los que no la cumplan saldrán por la salida falsa (salida de abajo). Otra forma de dirigir a los agentes por una salida u otra es con probabilidad. Para ello, se fija la probabilidad con la que el agente saldrá por la salida verdadera.

Siguiendo el mismo ejemplo que se viene estudiando en apartados anteriores, si las piezas pasan todas por una estación de torneado antes de salir del sistema, pero las piezas tipo 1 deben pasar antes por una estación de taladrado, habrá de usarse un módulo *SelectOutput* para dirigir a las piezas. Se impondrá la siguiente condición: si el atributo `TipoPieza` es igual a 1, la pieza irá a una estación de taladrado antes de la estación de torneado. En caso de que la condición no se cumpla, la pieza pasará directamente a la estación de torneado. La condición se definirá de la siguiente forma:

```
agent.TipoPieza==1
```

Esta expresión se reflejará en el campo *Condition* de las propiedades del módulo, que se activa al seleccionar la opción *If condition is true* de la propiedad del módulo *Select true output*.

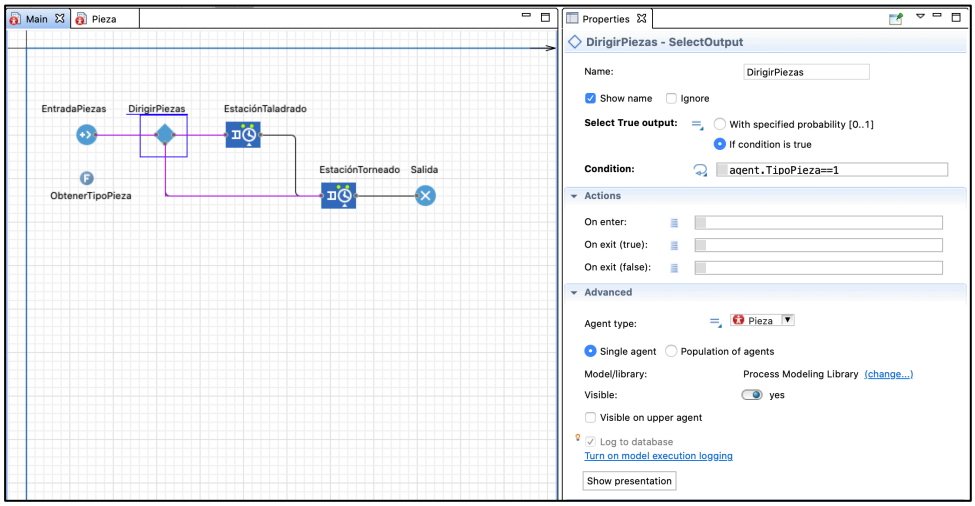


Figura 6–19. Módulo SelectOutput

En el fichero XML, al igual que pasó con el módulo *Source*, se habrá creado una nueva etiqueta `EmbeddedObject` dentro de la etiqueta `ActiveObjectClass` del agente `Main` y una etiqueta genérica `EmbeddedObjects`.

```
<EmbeddedObject>
    <Id>1595155054971</Id>
    <Name><![CDATA[DirigirPiezas]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[SelectOutput]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[SelectOutput]]></ClassName>
            <ItemName><![CDATA[1412336242931]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[conditionIsProbabilistic]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[false]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[condition]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[agent.TipoPieza==1]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[probability]]></Name>
        </Parameter>
        [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–20. XML de un módulo SelectOutput

6.3.5 Recursos

Los recursos representan a los operarios, las máquinas, equipos, herramientas, etc. Los recursos son los medios necesarios para llevar a cabo un proceso, éstos pueden estar libres o siendo utilizados por una estación de trabajo, por ejemplo.

Los recursos son modelados en AnyLogic mediante el módulo *Resource Pool* y pueden ser estáticos, móviles o portátiles.

Los objetos que solicitan un recurso se ponen en cola mediante un sistema FIFO, también puede estar basada en prioridades.

La capacidad del módulo *Resource Pool* es definida como el número de unidades de recurso existentes en el modelo. Esta capacidad puede ser definida, entre otras opciones, especificando directamente el número existente de unidades de recurso.

Continuando con el mismo ejemplo que en apartados anteriores y al sistema se le añaden, como recursos, un taladro y dos tornos. El taladro se trataría de un recurso portátil y los tornos serían estáticos.

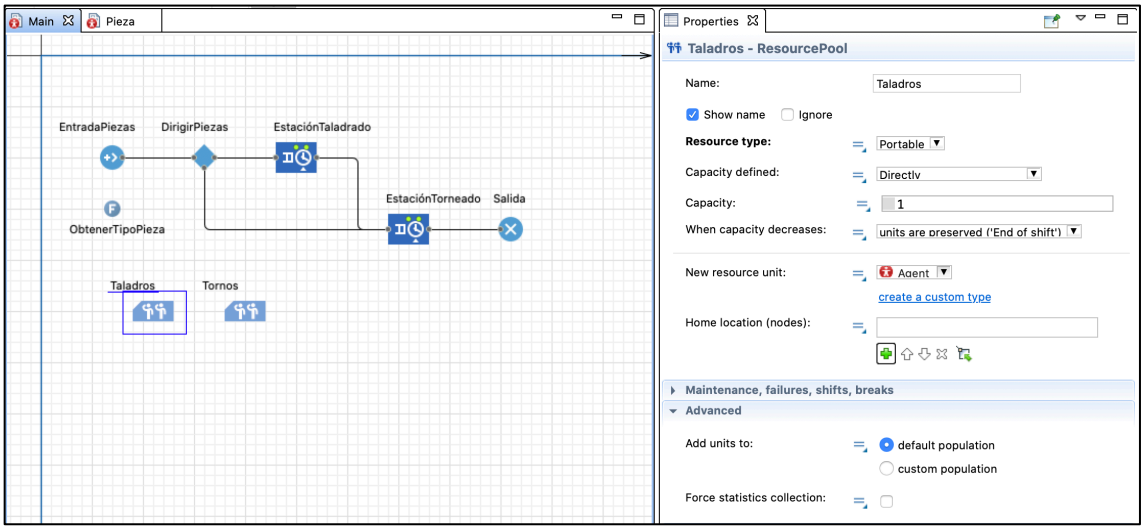


Figura 6–21. Módulo Resource Pool para recurso portátil

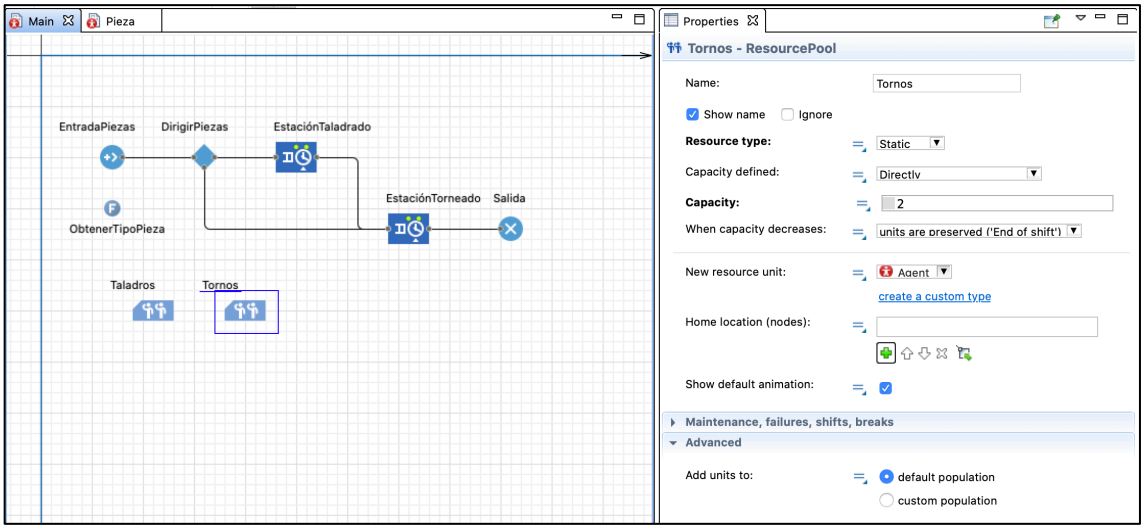


Figura 6–22. Módulo Resource Pool para recursos estáticos

Cada uno de estos módulos aparecen también en unas nuevas etiquetas `EmbeddedObject` en el fichero XML. Como anteriormente, estas etiquetas son creadas en una etiqueta genérica `EmbeddedObjects` dentro de la etiqueta `ActiveObjectClass` del agente `Main`.

```
<EmbeddedObject>
    <Id>1595172184803</Id>
    <Name><![CDATA[Tornos]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[ResourcePool]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[ResourcePool]]></ClassName>
            <ItemName><![CDATA[1412336243135]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[type]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.RESOURCE_STATIC]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[capacityDefinitionType]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[capacity]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[2]]></Code>
            </Value>
        </Parameter>
        [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–23. XML para recurso estático y con dos unidades de recurso

En el código XML se distingue lo siguiente:

- Etiqueta marcando el nombre del módulo.
- Etiqueta `ActiveObjectClass`, que activa un nuevo objeto.
- Etiqueta `GenericParameterSubsitute`, que escifica el nuevo objeto.
- Etiqueta de tipo `Parameter` con el tipo de recursos. Nótese que el código será el siguiente según el tipo:
  - Móvil: `self.RESOURCE_MOVING`. Esta es la opción por defecto de AnyLogic, por lo que puede dejarse la etiqueta correspondiente vacía.
  - Estático: `self.RESOURCE_STATIC`.
  - Portátil: `self.RESOURCE_PORTABLE`.
- Etiqueta de tipo `Parameter` con el método con el que se define la capacidad del recurso. En este caso, la capacidad de ambos recursos está definida directamente, que es el método por defecto en AnyLogic. Por ello, la etiqueta correspondiente no tiene código, aunque podría haberse reflejado de la siguiente forma:

```
<Parameter>
    <Name><![CDATA[capacityDefinitionType]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[self.CAPACITY_DIRECT]]></Code>
    </Value>
</Parameter>
```

Figura 6–24. Etiqueta Parameter para capacidad definida directamente de un módulo Resource Pool

- Etiqueta de tipo `Parameter` con la capacidad definida del recurso. Esta etiqueta no tendría código si la capacidad es definida directamente y su valor es 1, pues es el caso por defecto que contempla AnyLogic.

El código para el recurso del taladro sería similar al código de la Figura 6–23, teniendo en cuenta las observaciones anteriormente descritas.

6.3.6 Proceso con utilización de recursos

Una operación puede estar formada por una cola de espera y el proceso. Además, para realizar ese proceso la estación de trabajo necesitará unidades de recurso. Esta secuencia puede ser modelada en AnyLogic mediante el módulo *Service*. Este módulo servirá para modelar estaciones de trabajo, procesos en máquinas CNC o

movimiento de un puente grúa, entre otros muchos ejemplos.

El módulo *Service* sigue la siguiente secuencia: se apodera de un número determinado de unidades de recurso, retrasa al agente y libera las unidades de recurso incautadas. Retrasar al agente significa la realización del proceso en cuestión. Cuando el agente termina el proceso (el retraso) se liberan automáticamente aquellas unidades de recurso que fueron incautadas antes del retraso.

Este módulo tiene incorporada su propia cola, con su capacidad correspondiente. No obstante, la capacidad puede definirse como la máxima posible.

Los recursos para incautar pueden ser de un solo tipo o de varios, es decir, de la misma fuente o de diferentes fuentes. Por ejemplo, puede ser necesario solo máquinas CNC, o máquinas CNC y programadores. También habrá de especificarse el número de unidades de recurso necesarias de cada fuente para llevar a cabo la operación.

Avanzando en el ejemplo se viene describiendo, las estaciones de taladrado y torneado pueden ser modeladas con módulos *Service*. Se supone que la estación de taladrado incautará una unidad de recurso del recurso `Taladros`, tendrá la máxima capacidad de cola posible y el retraso del agente seguirá una distribución normal. La distribución normal será: `normal(0.5,2.0)` minutos.

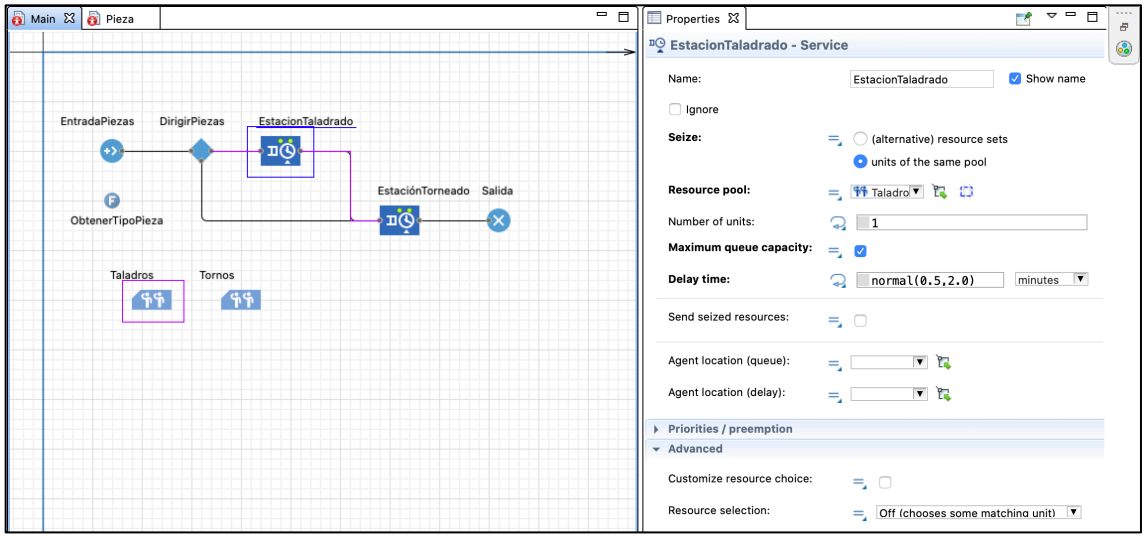


Figura 6–25. Módulo Service con capacidad máxima de cola y retraso según distribución normal

En el fichero XML aparecerá una nueva etiqueta `EmbeddedObject` para este módulo `EstacionTaladrado`.

Se distingue en el código lo siguiente:

- Etiqueta marcando el nombre del módulo.
- Etiqueta `ActiveObjectClass`, que activa un nuevo objeto.
- Etiqueta `GenericParameterSubsitute`, que especifica el nuevo objeto.
- Etiqueta `Parameter` para especificar que las unidades de recurso por incautar son de una misma fuente.
- Etiqueta `Parameter` para definir de qué fuente se incautan las unidades de recurso. Nótese que el parámetro posterior llamado `resourceQuantity` está vacío, pues las unidades a incautar son 1 taladro por agente a procesar y es el valor por defecto.
- Etiqueta `Parameter` especificando el retraso provocado al agente.

```
<EmbeddedObject>
    <Id>1595180042611</Id>
    <Name><![CDATA[EstacionTaladrado]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Service]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Service]]></ClassName>
            <ItemName><![CDATA[1412336243141]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[seizeFromOnePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceSets]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourcePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[Taladros]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceQuantity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[seizePolicy]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[queueCapacity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maximumCapacity]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[delayTime]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[normal(0.5,2.0)]]></Code>
                <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
        </Parameter>
        [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–26. XML módulo Service con capacidad máxima de cola y retraso según distribución normal

De igual forma, se puede proceder con la estación de torneado. No obstante, el proceso de torneado tendrá un tiempo u otro en función del tipo de pieza. En ese caso, el tiempo de retraso del agente en este módulo dependerá de un atributo, el atributo `TiempoTorno`, que tendrá un valor según el tipo de pieza que sea.

Las piezas de tipo 1 tendrán un tiempo de procesado según una distribución `normal(1.5,4.0)` minutos y las de tipo 2 tendrán una distribución `normal(2.0,10.0)` minutos.

El valor del atributo `TiempoTorno` puede definirse en las salidas de las piezas en el módulo `DirigirPiezas`. Si las piezas salen por la salida verdadera serán de tipo 1 y habrá de asignarle al atributo `TiempoTorno` la distribución normal anteriormente descrita. Por el contrario, si las piezas salen por la salida falsa serán de tipo 2 y habrá de asignarle al atributo `TiempoTorno` la otra distribución normal anteriormente definida. El módulo `DirigirPiezas` del modelo quedaría de la siguiente forma:

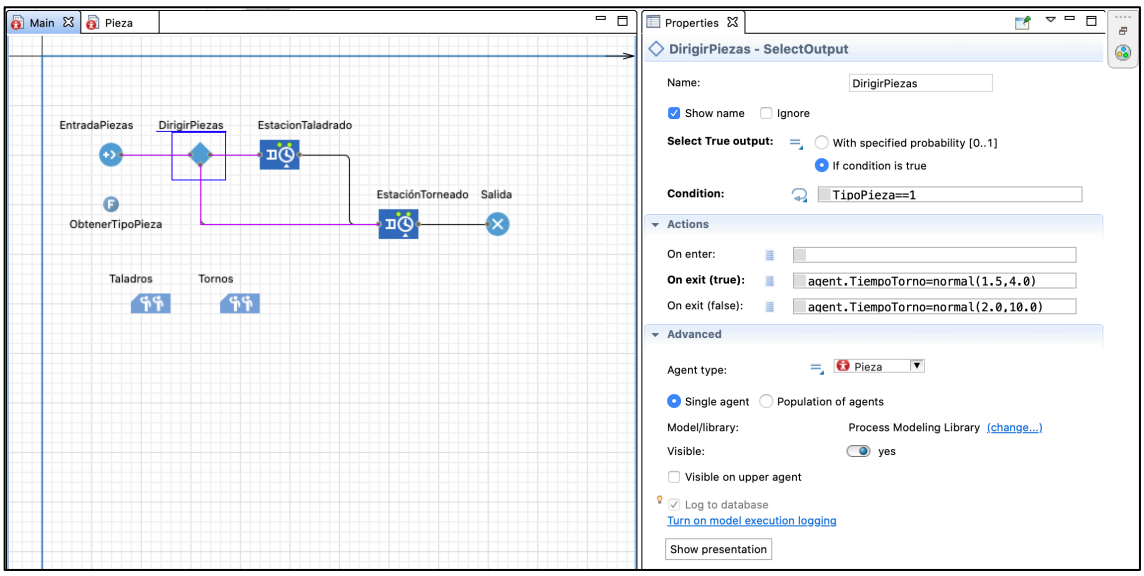


Figura 6–27. Asignación de valor a un atributo en módulo SelectOutput

En la etiqueta `EmbeddedObject` correspondiente al módulo `DirigirPiezas` del fichero XML se añadirían los siguientes parámetros:

```
<Parameter>
    <Name><![CDATA[onExitTrue]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[agent.TiempoTorno=normal(1.5,4.0)]]></Code>
    </Value>
</Parameter>
<Parameter>
    <Name><![CDATA[onExitFalse]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[agent.TiempoTorno=normal(2.0,10.0)]]></Code>
    </Value>
</Parameter>
```

Figura 6–28. XML asignación de valor a un atributo en módulo SelectOutput

Ahora, ya podría definirse la estación de torneado mediante un módulo *Service*, se nombrará `EstacionTorneado`. Esta estación incautará una unidad de recurso del recurso `Tornos` por cada agente, tendrá una capacidad de 80 piezas y el retraso del agente estará definido por el atributo `TiempoTorno`. La estación de torneado incautará una unidad de recurso del recurso `Tornos` por cada agente y la fuente de recursos `Tornos` tiene disponibles dos unidades de recurso, dos tornos. Sin embargo, para procesar un agente solo será necesario la utilización de un único torno.

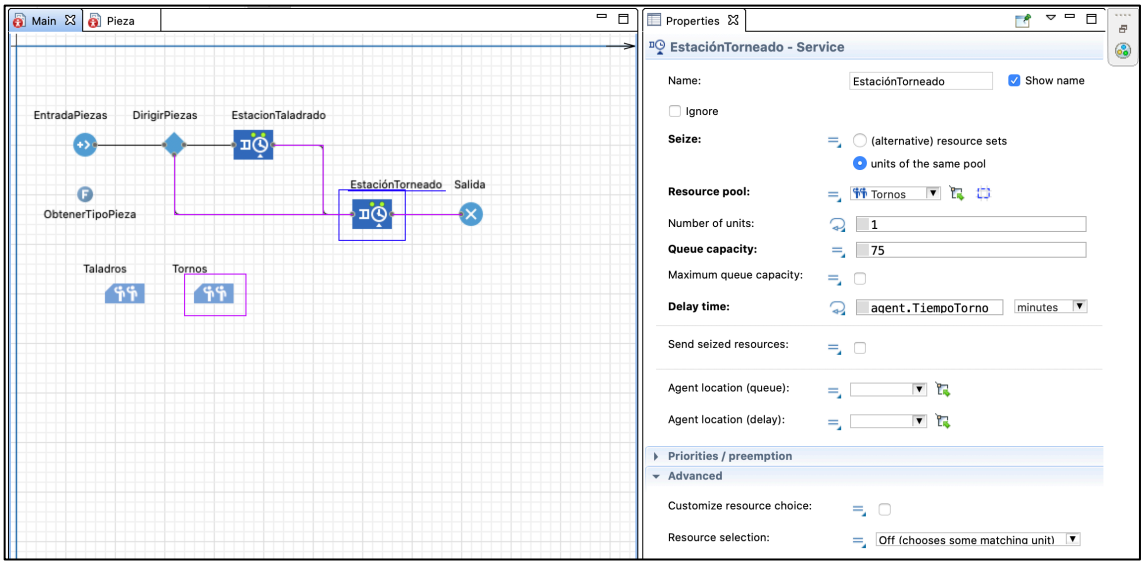


Figura 6–29. Módulo Service con capacidad definida y retraso según atributo

De igual forma que la estación de taladrado, esta estación creará una nueva etiqueta `EmbeddedObject` en el fichero XML. Esta etiqueta tendrá la misma estructura, se encontrarán diferencias en la etiqueta que defina el parámetro de tener una capacidad máxima impuesta.

```
<EmbeddedObject>
    <Id>1595156137187</Id>
    <Name><![CDATA[EstaciónTorneado]]></Name>
    <X>340</X><Y>150</Y>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Service]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Service]]></ClassName>
            <ItemName><![CDATA[1412336243141]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[seizeFromOnePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceSets]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourcePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[Tornos]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceQuantity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[seizePolicy]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[queueCapacity]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[75]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maximumCapacity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[delayTime]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[agent.TiempoTorno]]></Code>
                <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
        </Parameter>
        [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–30. XML módulo Service con capacidad definida y retraso según atributo

La funcionalidad del módulo *Service* puede ser sustituida por la secuencia de los módulos *Seize*, *Delay* y *Release*. En los siguientes tres apartados se verá en detalle estos tres módulos de AnyLogic.

6.3.7 Utilización de recursos

Cuando es necesario la utilización de recursos para realizar una operación, en primer lugar, habrá que incautar esos recursos. Para ello, AnyLogic dispone del módulo *Seize*, que se apodera de un número determinado de unidades de recurso de un módulo *Resource Pool* determinado. Cuando un agente llega a un módulo *Seize*, se solicita el recurso correspondiente y, una vez que el recurso ha sido otorgado, el agente abandona el módulo inmediatamente.

El módulo *Seize* cuenta con una cola propia donde los agentes esperan a los recursos. El recurso se solicita

para el primer agente de la cola. No obstante, la cola puede estar basada en prioridades. Por otro lado, la cola puede tener una capacidad definida o la capacidad máxima posible.

Como ejemplo: existe una línea productiva donde los agentes llegan al sistema y pasan por un módulo *Seize* para solicitar una máquina y son procesados por la mima. En ese caso, se tendrá un módulo *Resource Pool* llamado `Maquina1`, que tendrá únicamente una unidad de recurso: la máquina 1. El módulo *Seize* solicitará una unidad de recurso, tendrá la máxima capacidad de cola posible y esta cola no estará basada en prioridades.

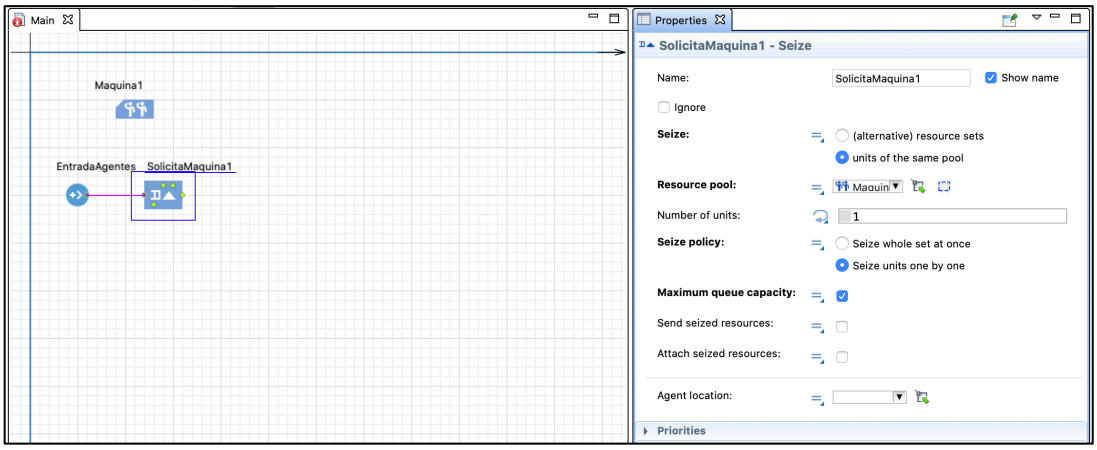


Figura 6–31. Módulo Seize con máxima capacidad de cola posible y sin prioridades

El módulo `SolictaMaquina1` se encontrará en su etiqueta `EmbeddedObject` correspondiente del fichero de XML. Como se viene viendo, estas etiquetas son creadas dentro de la etiqueta `ActiveObjectClass` del agente `Main` y una etiqueta genérica `EmbeddedObjects`.

```
<EmbeddedObject>
    <Id>1595265588679</Id>
    <Name><![CDATA[SolicitaMaquina1]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Seize]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Seize]]></ClassName>
            <ItemName><![CDATA[1412336243147]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[seizeFromOnePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceSets]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourcePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[Maquina1]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourceQuantity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[seizePolicy]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.SEIZE_UNITS_ONE_BY_ONE]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[capacity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maximumCapacity]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
      [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–32. XML módulo Seize con máxima capacidad de cola posible y sin prioridades

En el fichero XML, correspondiente al módulo `SolicitaMaquina1`, se puede destacar lo siguiente:

- Etiqueta marcando el nombre del módulo.
- Etiqueta `ActiveObjectClass`, que activa un nuevo objeto.
- Etiqueta `GenericParameterSubstitute`, que especifica el nuevo objeto.
- Etiqueta `Parameter` mostrando que las unidades de recursos serán incautadas de una misma fuente de recursos.
- Etiqueta `Parameter` indicando de qué fuente de recurso se deben incautar las unidades de recurso. Nótese que la siguiente etiqueta `Parameter`, correspondiente a la cantidad de unidades de recurso a incautar no tiene valor, pues solo se incautará una unidad de recurso y este es el valor por defecto.
- Etiqueta `Parameter` apuntando que la cola tiene la capacidad máxima de cola posible.

6.3.8 Proceso

La realización de un proceso puede modelarse como un retraso que se le provoca al agente por un periodo de tiempo determinado. Para ello, AnyLogic dispone del módulo *Delay*. El tiempo de retraso se evalúa dinámicamente, puede ser estocástico y puede depender del agente y de cualquier otra condición.

Este módulo puede tener capacidad para varios agentes. El módulo no permitirá que entren nuevos agentes hasta que no se completen los tiempos de retraso de los actuales agentes que están siendo procesados y la capacidad sea liberada.

Si se continúa el ejemplo del apartado anterior y se supone que los agentes deben ser procesados según una distribución `normal(0.5,4.0)` minutos, habría que insertar un módulo *Delay* para modelar este retraso. El módulo tendrá capacidad para un solo agente.

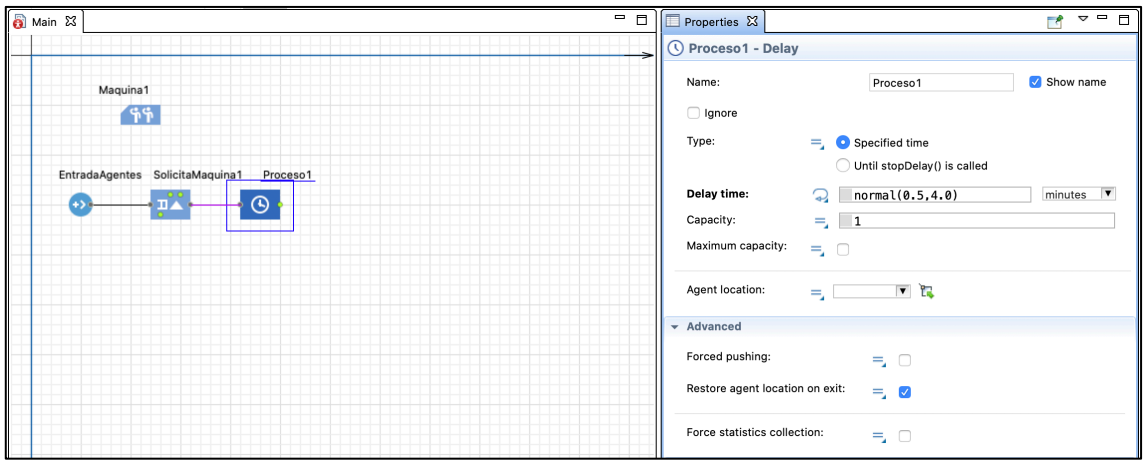


Figura 6–33. Módulo Delay para tiempo especificado y capacidad definida

Este módulo supondrá una nueva etiqueta `EmbeddedObject` en el fichero XML, ubicada de igual forma que en casos anteriores (véase Figura 6–36).

En el fichero XML se puede observar que hay dos etiquetas de parámetros vacíos: `type` y `capacity`. Ambas se encuentran vacía porque tienen los valores por defecto, pero podrían ser codificadas como sigue a continuación:

- Etiqueta de parámetro `type` cuando un módulo *Delay* termina cuando el tiempo de retraso especificado transcurre (campo *Specified time* en AnyLogic):

```
<Parameter>
    <Name><![CDATA[type]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[self.TIMEOUT]]></Code>
    </Value>
</Parameter>
```

Figura 6–34. Etiqueta de parámetro en fichero XML para módulo Delay de tipo tiempo especificado

- Etiqueta de parámetro `capacity` cuando un módulo *Delay* tiene capacidad para un solo agente (sirva esto también para saber cómo definir la capacidad de este módulo cuando no sea la unidad):

```
<Parameter>
    <Name><![CDATA[capacity]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
    </Value>
</Parameter>
```

Figura 6–35. Etiqueta de parámetro en fichero XML para módulo Delay con capacidad definida

```
<EmbeddedObject>
    <Id>1595267901850</Id>
    <Name><![CDATA[Proceso1]]></Name>
    [...]
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Delay]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Delay]]></ClassName>
            <ItemName><![CDATA[1412336242930]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[type]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[delayTime]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[normal(0.5,4.0)]]></Code>
                <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[capacity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maximumCapacity]]></Name>
        </Parameter>
        [...]
    </Parameters>
   [...]
</EmbeddedObject>
```

Figura 6–36. XML módulo Delay para tiempo especificado y capacidad definida

6.3.9 Liberación de recursos

Habrá que usarse el módulo *Release* para liberar las unidades de recurso incautadas previamente por un módulo *Seize*. La operación de liberar a los recursos conlleva tiempo cero. Los recursos incautados deben ser liberados antes de que el agente se elimine.

Los recursos móviles, tras su liberación, pueden volver a su ubicación de inicio (si no son incautados inmediatamente por otro agente) o permanecer donde están.

El módulo *Release* puede liberar recursos de las siguientes formas: todos los recursos incautados (de cualquier fuente), todos los recursos incautados por los módulos *Seize* dados, todos los recursos incautados de las fuentes dadas, recursos específicos (lista de fuentes) o cantidad de recursos especificada.

Siguiendo el ejemplo, habrá que añadir un módulo *Release* para liberar la máquina 1. Este módulo liberará una cantidad de recursos especificada, que será una unidad de la fuente de recursos `Maquina1`.

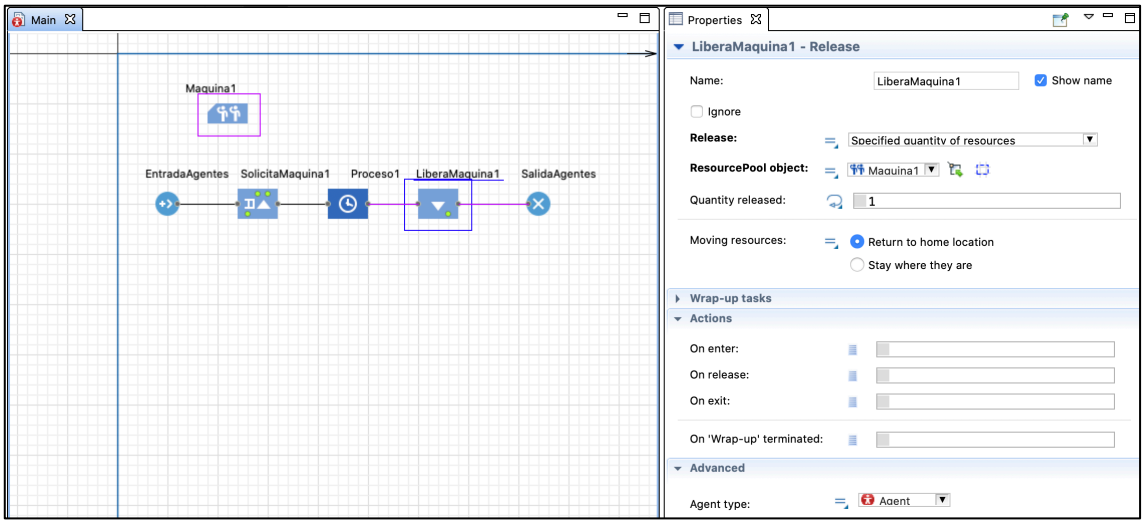


Figura 6–37. Módulo Release para liberación de una cantidad de recursos especificada

El fichero XML generaría la siguiente nueva etiqueta `EmbeddedObject`:

```
<EmbeddedObject>
    <Id>1595271447657</Id>
    <Name><![CDATA[LiberaMaquina1]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Release]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Release]]></ClassName>
            <ItemName><![CDATA[1412336243154]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[releaseMode]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.SPECIFIED_QUANTITY]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resourcePool]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[Maquina1]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[quantity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[resources]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[seizeBlocks]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[movingGoHome]]></Name>
        </Parameter>
        [...]
    </Parameters>
   [...]
</EmbeddedObject>
```

Figura 6–38. XML módulo Release para liberación de una cantidad de recursos especificada

A continuación, se detallan algunos cambios que se producirían en el fichero XML en función de las propiedades que se definan en AnyLogic para este módulo *Release*.

Si la cantidad a liberar fuera diferente a la unidad, que es el valor por defecto, la etiqueta correspondiente sería como muestra la figura de a continuación (se ha realizado la etiqueta para cantidad 1 para no alterar el modelo, sería de forma idéntica para otro número a liberar):

```
<Parameter>
    <Name><![CDATA[quantity]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[1]]></Code>
    </Value>
</Parameter>
```

Figura 6–39. Etiqueta de parámetro en XML para módulo Release con cantidad a liberar

Por otro lado, el recurso puede que no vuelva a su lugar de inicio. En ese caso la etiqueta de parámetro `movingGoHome` quedaría de la siguiente forma:

```
<Parameter>
    <Name><![CDATA[movingGoHome]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[false]]></Code>
    </Value>
</Parameter>
```

Figura 6–40. Etiqueta de parámetro en XML para módulo Release y los recursos no van a su incio

6.3.10 Cola

Una cola (o búfer) es una estación de almacenamiento en un proceso productivo. Estos espacios de almacenamiento pueden encontrarse, por ejemplo, a la entrada y salida de una máquina.

Las colas son modeladas en AnyLogic mediante el módulo *Queue*. En este módulo los agentes esperan a ser aceptados por los siguientes objetos en el flujo del proceso, también pueden encontrarse estos módulos como un almacenamiento de propósito general para los agentes.

Las colas pueden tener un orden FIFO, LIFO o basada en prioridades. La prioridad puede almacenarse explícitamente en el agente o calcularse en función de las propiedades del agente y las condiciones externas. Una cola prioritaria siempre acepta un agente entrante, evalúa su prioridad y la coloca en la posición correspondiente en la cola.

La capacidad de la cola puede ser cambiada dinámicamente.

Continuando el ejemplo que se venía describiendo, se puede añadir un módulo *Queue* antes de la salida de los agentes del sistema. Esta cola tendrá la máxima capacidad posible, un orden LIFO y los agentes saldrán de este módulo tras 3 minutos.

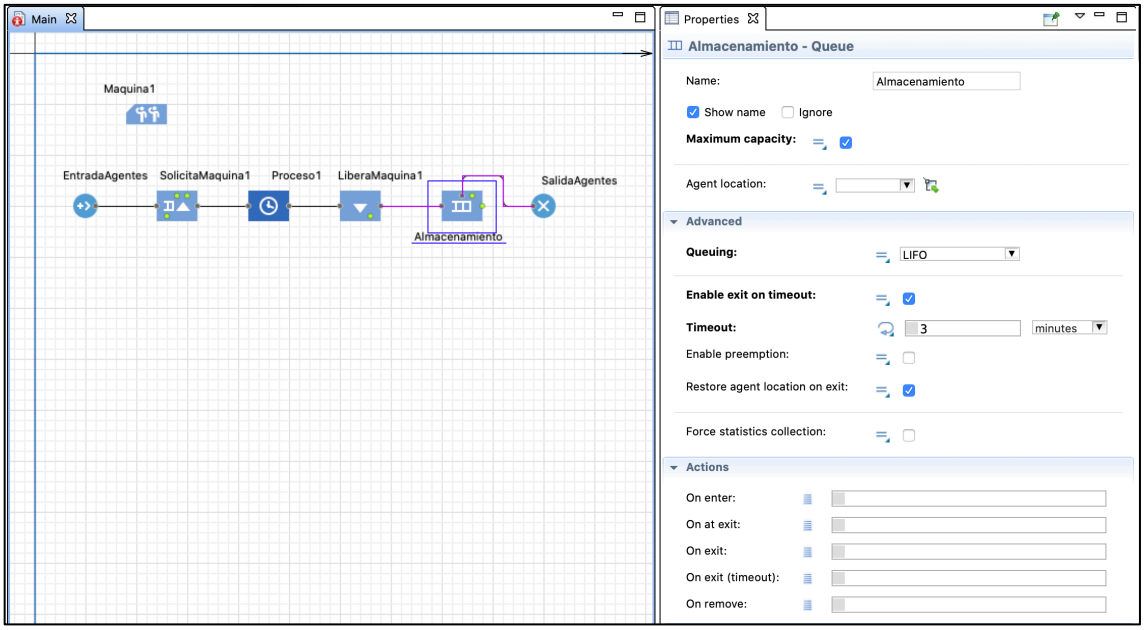


Figura 6–41. Módulo Queue con capacidad máxima posible, LIFO y tiempo

En el fichero XML aparecerá una nueva etiqueta `EmbeddedObject` para este módulo *Queue* denominado

`Almacenamiento`. Como anteriormente, estas etiquetas son creadas dentro de la etiqueta `ActiveObjectClass` del agente `Main` y una etiqueta genérica `EmbeddedObjects`.

Se podrá observar en el fichero XML las etiquetas correspondientes a las propiedades definidas. Estas etiquetas aparecerán como etiquetas `Parameter`. Se encontrarán las etiquetas para las siguientes propiedades definidas: máxima capacidad posible, orden LIFO y tiempo de almacenamiento definido.

```
<EmbeddedObject>
    <Id>1595349941896</Id>
    <Name><![CDATA[Almacenamiento]]></Name>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Queue]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Queue]]></ClassName>
            <ItemName><![CDATA[1412336242932]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[capacity]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[maximumCapacity]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        [...]
        <Parameter>
            <Name><![CDATA[queuing]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.QUEUING_LIFO]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[priority]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[comparison]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[enableTimeout]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[true]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[timeout]]></Name>
            <Value Class="CodeUnitValue">
                <Code><![CDATA[3]]></Code>
                <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
            </Value>
        </Parameter>
        [...]
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–42. XML módulo Queue con capacidad máxima posible, LIFO y tiempo

En caso de que la capacidad definida no fuera la máxima posible, sino una capacidad máxima de 150 agentes, el código correspondiente en el fichero XML sería el siguiente:

```
<Parameter>
    <Name><![CDATA[capacity]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[150]]></Code>
    </Value>
</Parameter>
```

Figura 6–43. XML para capacidad definida en módulo Queue

Otra alternativa que se puede encontrar en el módulo *Queue* es que el orden sea FIFO. En ese caso, la etiqueta

correspondiente al orden de la cola, que es la etiqueta de parámetro `queuing`, aparecería con el código `self.QUEING_FIFO`.

6.3.11 Bloqueo

El flujo de los agentes puede ser bloqueado. El bloqueo puede modelarse en AnyLogic mediante el módulo *Hold*.

Los agentes bloqueados no quedan dentro del módulo *Hold*, quedan dentro del módulo anterior. Los agentes pasan tiempo cero en este módulo. Existen tres formas diferentes para bloquear: de forma manual, tras un número especificado de agentes o mediante una condición personalizada.

Se ha tener en cuenta que en algún momento del diagrama de flujo habrá que desbloquear el flujo. En el caso de las formas manuales y un número especificado de agentes, habrá de llamarse a la función `unblock()`. Para el caso de condiciones, el módulo *Hold* evalúa las condiciones cuando un agente llega a su entrada, pero si el agente queda bloqueado en el módulo anterior, las condiciones no volverán a evaluarse mientras no se llame a la función `recalculateConditions()`. Estas funciones son llamadas de la siguiente forma: *`NombreModuloHold`*`.recalculateConditions()`, de igual forma para la función `unblock()`.

Como ejemplo: existe un sistema compuesto por dos máquinas en serie y la máquina 2 tiene cola con capacidad limitada: 3 unidades. Así, cuando se prevé que se alcanza el límite, la máquina 1 se bloquea. Las piezas que se van a bloquear van a un espacio o búfer con capacidad de una unidad. El modelo sería el siguiente:

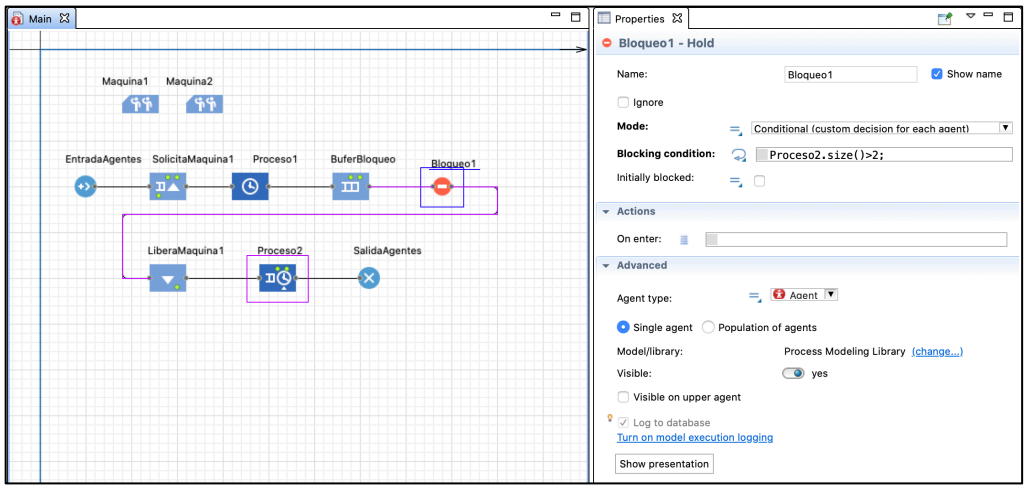


Figura 6–44. Módulo Hold modo condicional

La condición que se ha establecido a evaluar en AnyLogic ha sido:

```
Proceso2.size()>2;
```

Con esta condición, la máquina 1 quedará bloqueada cuando el módulo `Proceso2`, correspondiente a la máquina 2, contenga más de dos agentes en su interior: dos agentes esperando en cola y uno procesándose. La función correspondiente en AnyLogic para contar el número de agentes que contiene un módulo es: *`NombreModulo`*`.size()`.

Como se indicaba anteriormente, será necesario llamar a la función `recalculateConditions()` para que la condición de bloqueo sea nuevamente evaluada si existe un agente bloqueado en el módulo `BuferBloqueo` y la máquina 2 ya no está ociosa. Por ello, la llamada a esta función habrá de hacerse cada vez que un agente abandone la máquina 2, pues se verá disminuido el número de agentes en el módulo `Proceso2`. En el visor de acciones de las propiedades de este módulo, a la salida (campo *On exit*), se llamará a la función de la siguiente forma:

```
Bloqueo1.recalculateConditions();
```

El módulo *Hold* creará en el fichero XML una nueva etiqueta `EmbeddedObject`. Su código es de la siguiente forma:

```
<EmbeddedObject>
    <Id>1595355153280</Id>
    [...]
    <ActiveObjectClass>
        <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
        <ClassName><![CDATA[Hold]]></ClassName>
    </ActiveObjectClass>
    <GenericParameterSubstitute>
        <GenericParameterSubstituteReference>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Hold]]></ClassName>
            <ItemName><![CDATA[1412336242940]]></ItemName>
        </GenericParameterSubstituteReference>
    </GenericParameterSubstitute>
    <Parameters>
        <Parameter>
            <Name><![CDATA[mode]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[self.CONDITIONAL]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[nEntitiesForSelfBlock]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[condition]]></Name>
            <Value Class="CodeValue">
                <Code><![CDATA[Proceso2.size()>2;]]></Code>
            </Value>
        </Parameter>
        <Parameter>
            <Name><![CDATA[initiallyBlocked]]></Name>
        </Parameter>
        <Parameter>
            <Name><![CDATA[onEnter]]></Name>
        </Parameter>
    </Parameters>
    [...]
</EmbeddedObject>
```

Figura 6–45. XML módulo Hold modo condicional

Por otro lado, la llamada a la función para recalcular las condiciones de bloqueo es como aparece en la siguiente figura. Esta llamada aparecerá en la etiqueta `EmbeddedObject` correspondiente al módulo `Proceso2`.

```
<Parameter>
    <Name><![CDATA[onExit]]></Name>
    <Value Class="CodeValue">
        <Code><![CDATA[Bloqueo1.recalculateConditions();]]></Code>
    </Value>
</Parameter>
```

Figura 6–46. Llamada a la función recalculateConditions a la salida de un módulo

Se ha visto un ejemplo donde la forma de bloqueo es mediante una condición personalizada. Si se supone que el bloqueo actúa tras un número especificado de agentes y, además, el bloqueo está actuando al iniciar el modelo, los cambios en el fichero XML serían los siguientes:

```
<Parameters>
    <Parameter>
        <Name><![CDATA[mode]]></Name>
        <Value Class="CodeValue">
            <Code><![CDATA[self.BLOCK_AFTER_N_ENTITIES]]></Code>
        </Value>
    </Parameter>
    <Parameter>
        <Name><![CDATA[nEntitiesForSelfBlock]]></Name>
        <Value Class="CodeValue">
            <Code><![CDATA[10]]></Code>
        </Value>
    </Parameter>
    <Parameter>
        <Name><![CDATA[condition]]></Name>
    </Parameter>
    <Parameter>
        <Name><![CDATA[initiallyBlocked]]></Name>
        <Value Class="CodeValue">
            <Code><![CDATA[true]]></Code>
        </Value>
    </Parameter>
    <Parameter>
        <Name><![CDATA[onEnter]]></Name>
    </Parameter>
</Parameters>
```

Figura 6–47. XML parámetros módulo Hold que actúa tras un número especificado de agentes

La forma de desbloqueo será manual. Por ello, en algún momento habrá de llamarse a la función `Bloqueo1.unblock()`.

6.3.12 Variables

Las variables almacenan datos del modelo, pueden ser datos predefinidos necesarios para el modelo o valores que se van almacenando a lo largo de la simulación. Las variables pueden ser de diferentes tipos, desde valores enteros o flotantes, hasta cadenas de caracteres. El bloque *Variable* de la Agent Palette es el encargado de modelar las variables.

Volviendo al primer ejemplo que se planteó, el cual tenía dos tipos de piezas y en función del tipo de pieza la estación de torneado tenía un tiempo de procesado u otro, se puede tener una variable con un vector de flotantes con los tiempos de procesado definidos. En ese caso, el atributo `TiempoTorno` no será necesario y se puede definir la siguiente variable:

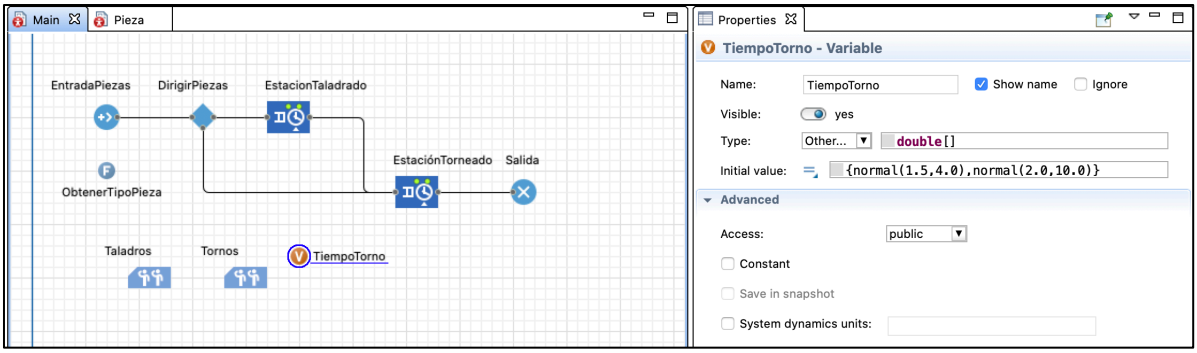


Figura 6–48. Módulo Variable con datos predefinidos para la simulación

La posición 0 del vector da el tiempo de procesado de las piezas tipo 1 y la posición 1 del vector da el tiempo de procesado de las piezas tipo 2. Ahora, en el módulo `EstacionTorneado` el tiempo de procesado no dependerá del atributo `TiempoTorno`, sino de la variable `TiempoTorno` y se hará la siguiente llamada en el campo *Delay time* de la ventana de propiedades:

```
TiempoTorno[agent.TipoPieza-1]
```

Las variables son creadas en el fichero XML en la etiqueta `Variables`. Esta es una etiqueta genérica que contiene unas etiquetas `Variable` por cada variable del modelo. La etiqueta `Variables` se encuentra en la etiqueta `ActiveObjectClass` del agente `Main`.

La estructura del fichero XML para este tipo de bloque es la siguiente:

```
<Variables>
    <Variable Class="PlainVariable">
        <Id>1595521649583</Id>
        <Name><![CDATA[TiempoTorno]]></Name>
        [...]
        <Properties SaveInSnapshot="true" Constant="false" AccessType="public" StaticVariable="false">
            <Type><![CDATA[double[] ]]></Type>
            <InitialValue Class="CodeValue">
                <Code><![CDATA[{normal(1.5,4.0),normal(2.0,10.0)}]]></Code>
            </InitialValue>
        </Properties>
    </Variable>
</Variables>
```

Figura 6–49. XML módulo Variable con datos predefinidos para la simulación

La llamada a esta variable en el módulo `EstacionTorneado` quedaría en la siguiente etiqueta de parámetro:

```
<Parameter>
    <Name><![CDATA[delayTime]]></Name>
    <Value Class="CodeUnitValue">
        <Code><![CDATA[TiempoTorno[agent.TipoPieza-1]]]></Code>
        <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
    </Value>
</Parameter>
```

Figura 6–50. Llamada a una variable en módulo Service

7 APLICACIÓN

En este apartado se realizará la aplicación de las técnicas y metodologías que en este proyecto se han presentado y estudiado. Se va a definir una estructura PPR de una línea de ensamblado simple y se realizará una instanciación de una ontología. Asimismo, la estructura PPR será transformada en un modelo de simulación y ejecutada en el software de simulación. Esta aplicación estará fundamentada en las herramientas que en apartados anteriores se han explicado: ontologías, MBSE, modelado de simulación, ontología para el modelado de simulación de eventos discretos, etc.

7.1 Definición de la ontología PPR

En primer lugar, se definirá una estructura PPR que represente los productos, procesos y recursos de la estructura de fabricación, así como las relaciones entre ellos. En este caso, la estructura de fabricación se trata de una línea de ensamblado. Esta estructura será diseñada mediante una ontología para que quede diseñada adecuadamente y resulte ser unívoca.

El modelo estará formado por clases y subclases que poseerán una serie de atributos. Las clases se relacionarán con otras mediante unas propiedades de objeto, que representarán la interacción entre clases. Las interacciones pueden establecerse como propiedades de dominio o rango, donde el rango sería la clase propiedad de otra principal (dominio).

La siguiente figura muestra la representación gráfica de la ontología PPR:

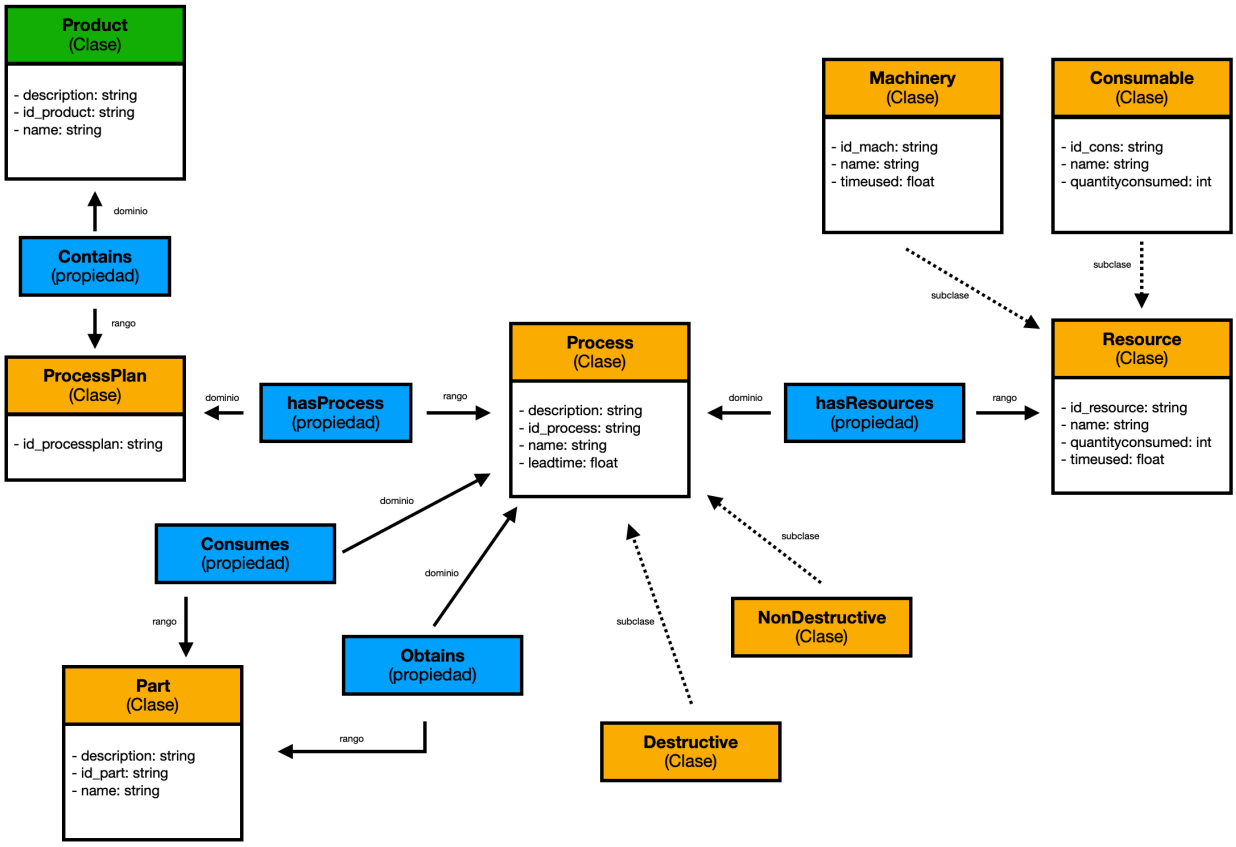


Figura 7–1. Arquitectura PPR de una línea de ensamblado

En el diagrama se puede observar que la clase de producto contiene al resto. El producto, que es el elemento final, será el conjunto ensamblado. Un producto tendrá asociado un plan de procesos, que estará formado por un conjunto de procesos necesarios para conseguir el conjunto ensamblado. En este caso, los procesos pueden ser de dos tipos diferentes, identificados mediante subclases. Los tipos de procesos que se han definido son: destructivos (por ejemplo, un torneado) y no destructivos (por ejemplo, un remachado). Asimismo, los procesos usan uno o varios recursos para ejecutarse. También en este caso, los recursos pueden ser de dos tipos: maquinaria o consumibles (bulones, remaches…).

La aplicación usada para la definición de la ontología es Protégé, una herramienta de software libre que permite al usuario construir modelos usando ontologías. Protégé soporta lenguajes como OWL y RDF-S.

El uso de Protégé es muy intuitivo. A continuación, se van a ir desarrollando los diferentes módulos de este software para conseguir la definición de la ontología (aplicado a la estructura de fabricación anteriormente presentada).

En el menú principal, en la opción de *Entities* se encuentran las distintas pestañas para especificar clases, propiedades, relaciones, atributos y anotaciones, así como la opción para la instanciación.

Clases

Las clases se crean en la pestaña *Classes*. Sobre la clase genérica `owl:Thing`, se añaden nuevas clases clicando en el icono *Add subclass*. Este icono también permite crear subclases que se hayan creado.

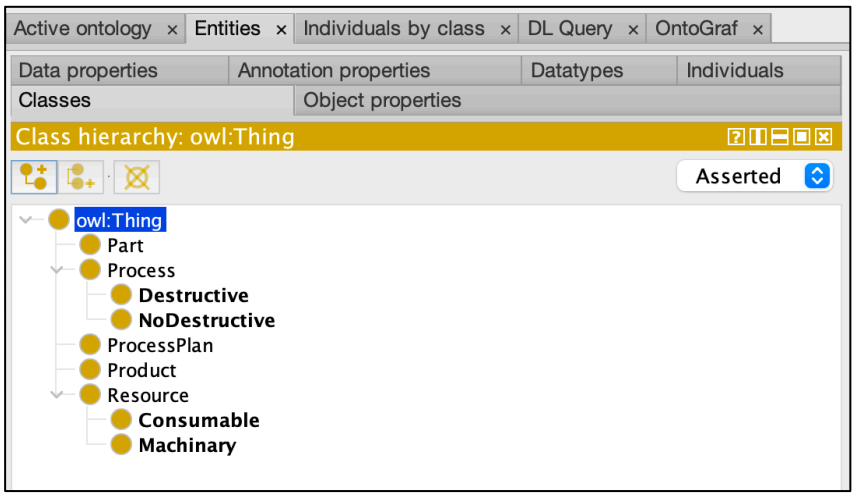


Figura 7–2. Pestaña de clases

En la ventana de la derecha aparecen las opciones de anotaciones y descripción de la clase, que permite establecer relaciones semánticas entre clases.

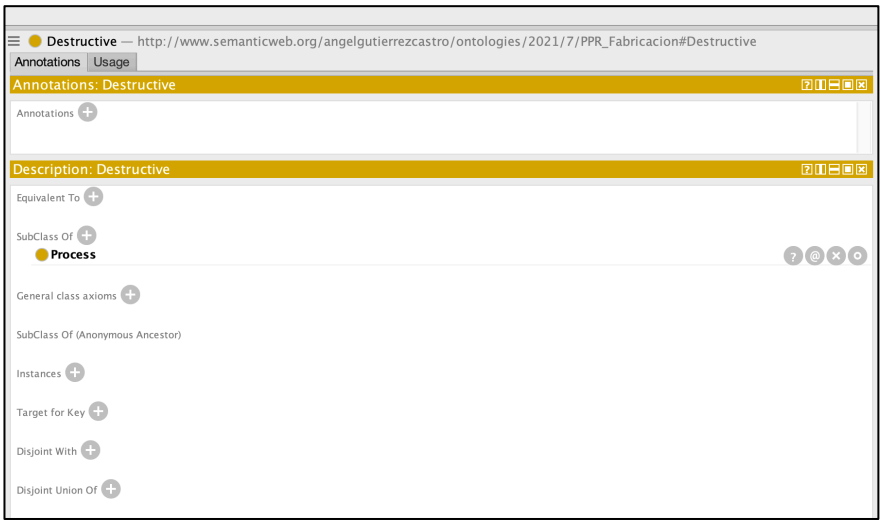


Figura 7–3. Ventana de propiedades de una clase

Propiedades

Las relaciones entre clases se realizan mediante propiedades de objeto, en la pestaña *Object properties*. Sobre el nodo raíz `owl:topObjectProperty`, se pueden crear tantas propiedades como se quieran clicando en el icono *Add sub property*.

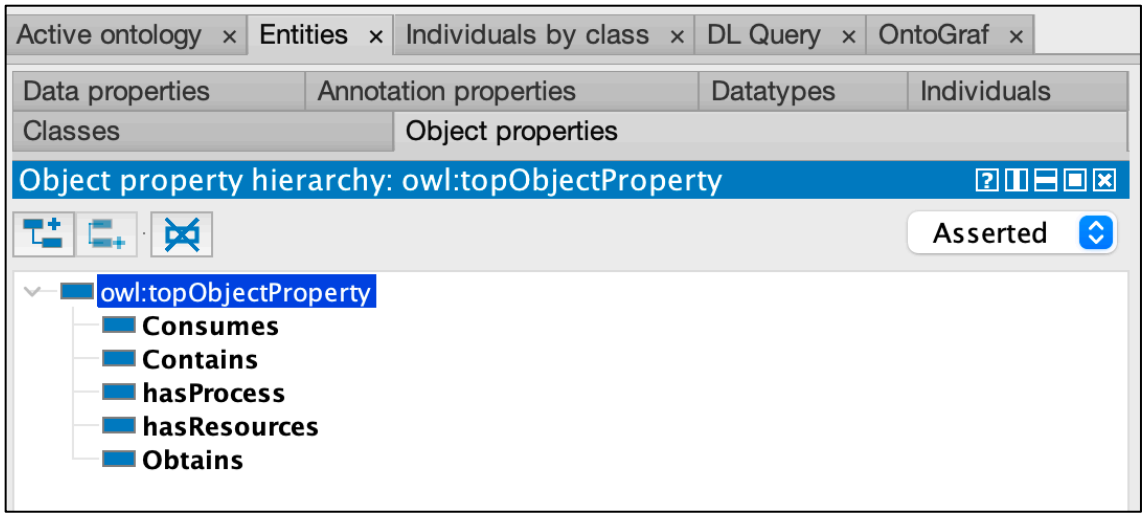


Figura 7–4. Pestaña de propiedades de objeto

Una vez creada una propiedad, en su ventana de la derecha *Description*, se habrá de establecer la información de dicha propiedad, entre ella, el dominio y rango.

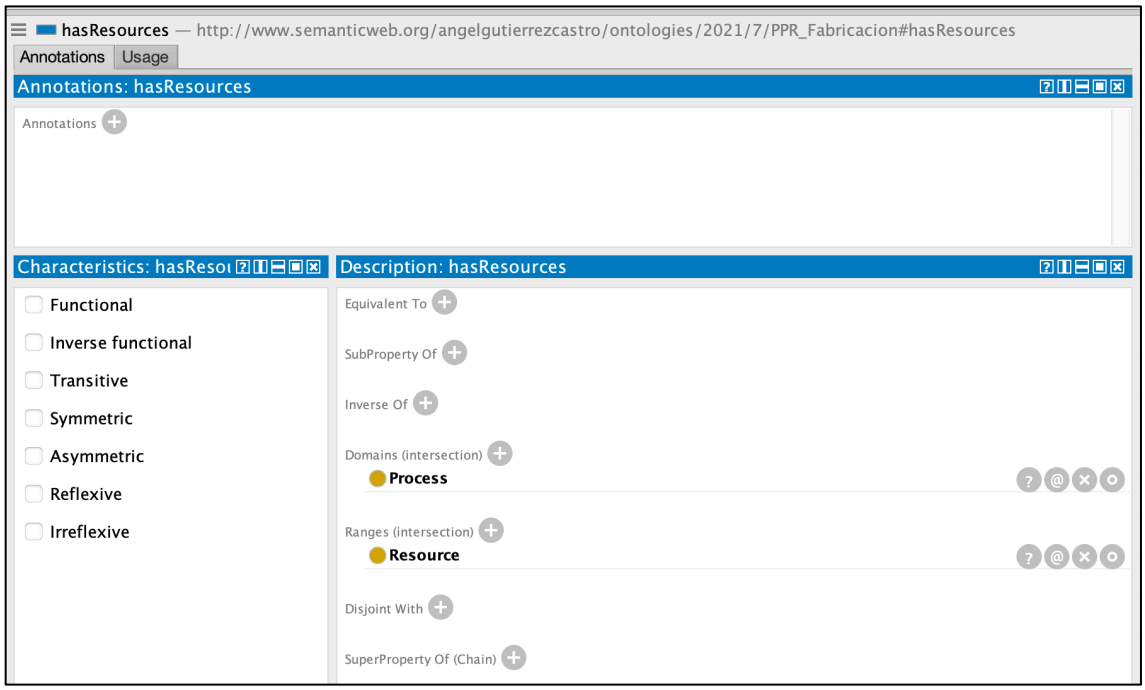


Figura 7–5. Ventana de descripción de las propiedades objeto

Atributos

Cada clase tiene una serie de atributos que indicarán la información que hay que proporcionar para su definición. Los atributos son definidos en la pestaña *Data properties*. Sobre el dato genérico `owl:topDataProperty`, se definen los diferentes atributos clicando en el icono *Add sub property*.

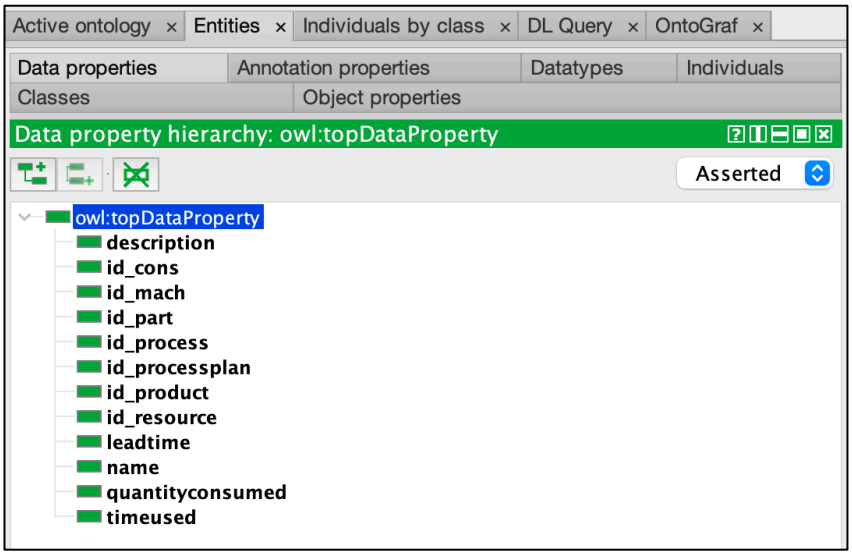


Figura 7–6. Pestaña de atributos

En la ventana de la derecha, para cada atributo se establece el tipo de dato (rango) y las clases a las que está asignado (dominio).

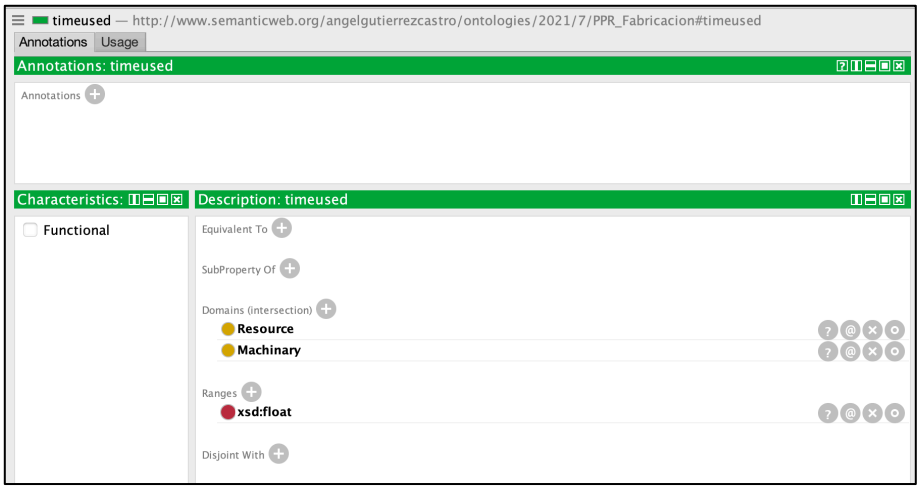


Figura 7–7. Ventana de descripción de atributos

Por otro lado, se encuentra la característica *Functional*, la cual indicará que un individuo solo puede tener ese atributo una vez. Por ejemplo, un producto solo tendrá un `id_product`. En el caso que se está tratando, todos los atributos definidos son funcionales.

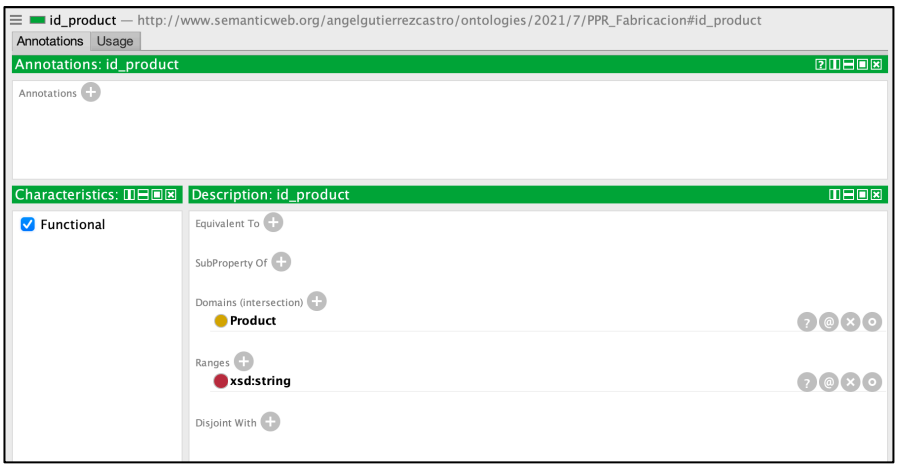


Figura 7–8. Característica funcional de atributos

Protégé no permite construir una ontología de forma gráfica, como sí permite por ejemplo el software Enterprise Architect. No obstante, una vez que la ontología es construida en Protégé, el software permite visualizar un gráfico de la ontología.

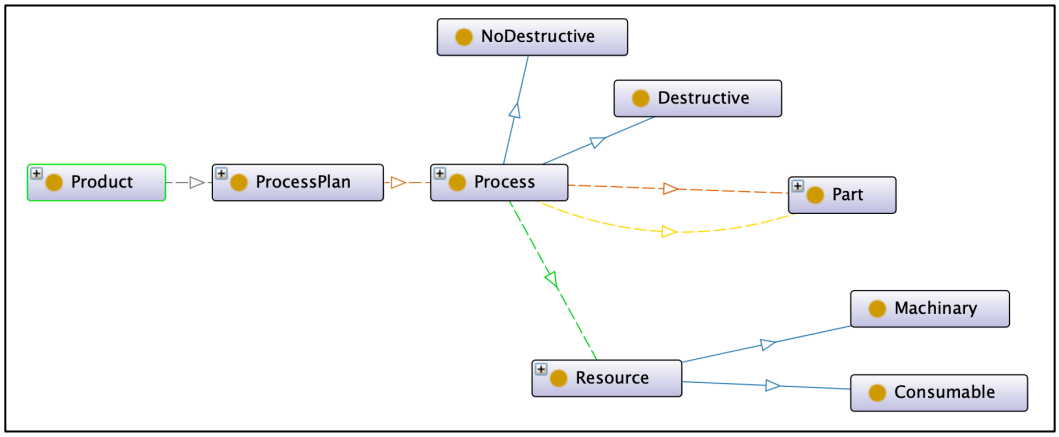


Figura 7–9. Gráfico de la ontología en Protégé

La herramienta Protégé permite guardar la ontología en diferentes formatos, entre ellos XML. La ontología construida se puede exportar a un fichero XML. Este fichero comenzará con la declaración XML y como hijos de la raíz se encontrarán tres partes bien diferenciadas:

1. Propiedades de objeto.
2. Atributos.
3. Clases.

En las siguientes figuras se pueden encontrar extractos de la exportación a XML.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/untitled-ontology-5#"
     xml:base="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/untitled-ontology-5"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion"/>
```

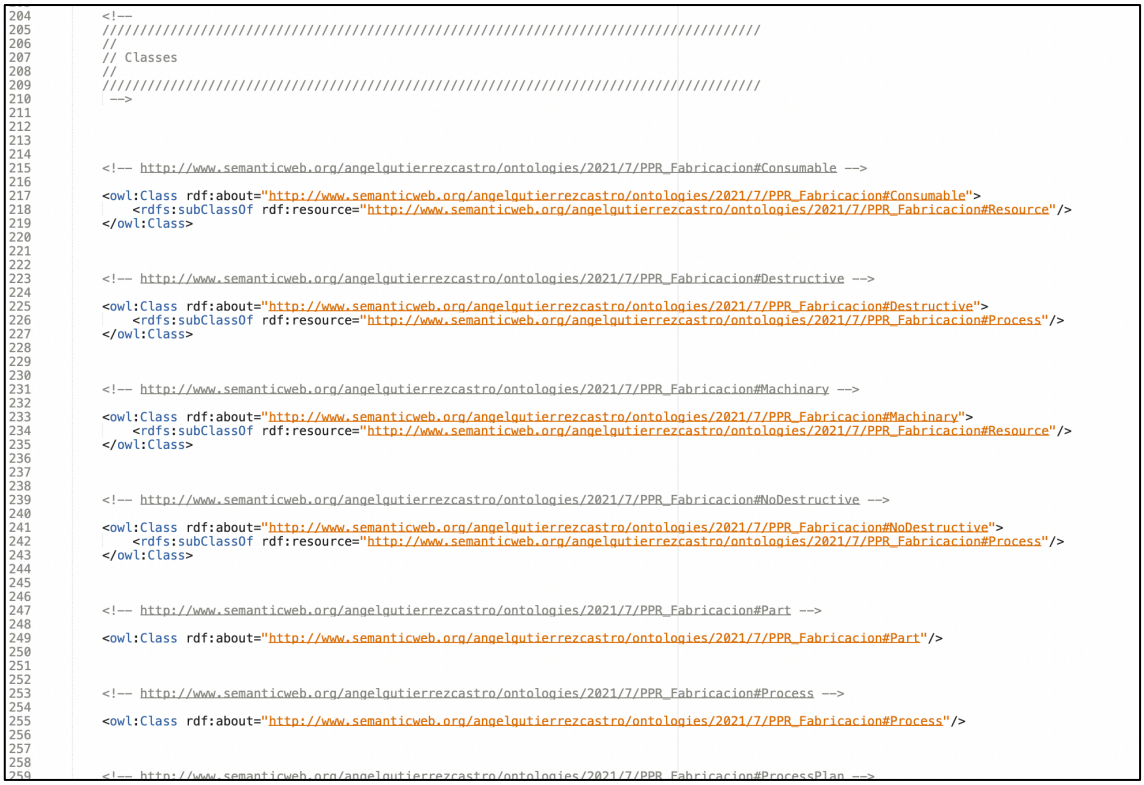
Figura 7–10. Declaración XML

```
    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Consumes -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Consumes">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Part"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Contains -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Contains">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Product"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#ProcessPlan"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Obtains -->

    <owl:ObjectProperty...>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#hasProcess -->

    <owl:ObjectProperty...>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#hasResources -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#hasResources">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Resource"/>
    </owl:ObjectProperty>
```

Figura 7–11. Propiedades de objeto en XML

```
    <!--
    ////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#description -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#description">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Part"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Product"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_cons -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_cons">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Consumable"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_mach -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_mach">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Machinary"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_part -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_part">
        <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Part"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:DatatypeProperty>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_process -->

    <owl:DatatypeProperty rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#id_process">
```

Figura 7–12. Atributos en XML



```
    <!--
    ////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Consumable -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Consumable">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Resource"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Destructive -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Destructive">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Machinary -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Machinary">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Resource"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#NoDestructive -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#NoDestructive">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>
    </owl:Class>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Part -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Part"/>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process -->

    <owl:Class rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Process"/>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#ProcessPlan -->
```

Figura 7–13. Clases en XML

7.2 Instanciación de la ontología

Definida la ontología, se puede realizar una instanciación (ejemplo) de la estructura definida, es decir, se

pueden generar entidades de los tipos de clases que se han creado y se le pueden indicar los valores de sus atributos.

Se ha tomado como ejemplo una simple línea de ensamblado en la que se obtiene un tipo de conjunto, tras un plan de procesos que incluye un torneado y un remachado. En las siguientes tablas se detallan los datos de partida del ejemplo en base a la estructura ontológica.

Tabla 7–1. Instanciación de clases

Product	**CE_1** - description: Conjunto Ensamblado 1 - id_product: CE_1 - name: CONJUNTO		
ProcessPlan	**Montaje_1** - id_processplan: Montaje_1		
Part	**1** - description: Pieza tipo 1 - id_part: 1 - name: PIEZA	**1T1** - description: Pieza tipo 1 tras taladrado 1 - id_part: 1T1 - name: PIEZA	**1T1R1** - description: Pieza tipo 1 tras taladrado 1 y remachado 1 - id_part: 1T1R1 - name: PIEZA
Process	**TAL_1** *(subclase Destructive)* - description: Taladrado de piezas 1 - id_process: TAL_1 - name: TALADRADO - leadtime: 5.0 (min)	**REM_1** *(subclase NonDestructive)* - description: Remachado de piezas 1 - id_process: REM_1 - name: REMACHADO - leadtime: 15.0 (min)	

Resource	**M1** *(subclase Machinary)* - id_mach: M1 - name: TALADRADORA - timeused: 5.0 (min)	**M2** *(subclase Machinary)* - id_mach: M2 - name: REMACHADORA - timeused: 15.0 (min)	**RIVET** *(subclase Consumable)* - id_mach: RIVET - name: REMACHE - quantityconsumed: 20 (uds)

Tabla 7–2. Propiedades de objeto de la instanciación

Propiedad	Dominio	Rango
Contains	CE_1	Montaje_1
hasProcess	Montaje_1	TAL_1
hasResources	TAL_1	M1
Consumes	TAL_1	1
Obtains	TAL_1	1T1
hasProcess	Montaje_1	REM_1
hasResources	REM_1	M2
hasResources	REM_1	RIVET
Consumes	REM_1	1T1
Obtains	REM_1	1T1R1

La instanciación en el software Protégé se realiza en la pestaña *Individuals*. Para crear una instancia hay que clicar sobre el icono *Add individual* y aparecerá una ventana para nombrar la instancia que se va a realizar. En el ejemplo que se está desarrollando el nombre de las instancias se ha establecido con el mismo ID que se ha dado a la clase.

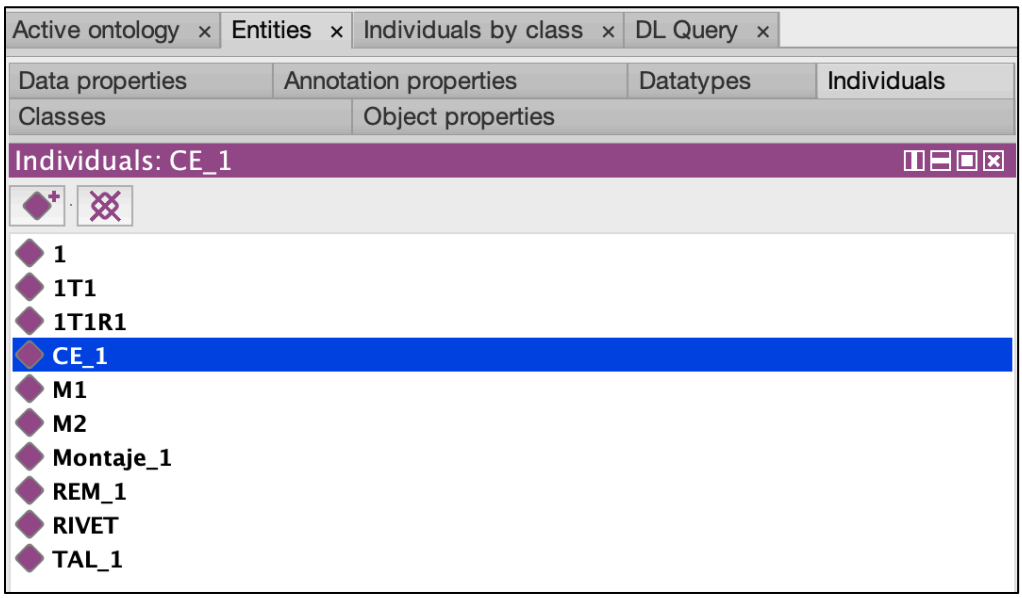


Figura 7–14. Creación de instancias

Una vez que se ha creado una instancia, en la ventana de la derecha se encuentran las opciones *Description* y *Property assertions*. En la opción *Description*, en *Types*, se indicará la clase de la instancia.

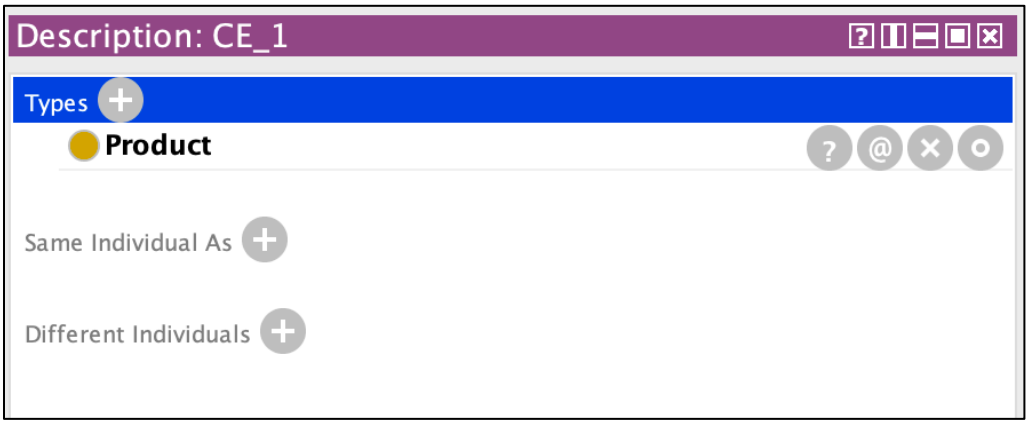


Figura 7–15. Asignación del tipo de clase en la instancia

Por otro lado, en la opción *Property assertions* se caracterizarán los valores de los atributos de la clase y las relaciones entre clases. Los valores de los atributos se indican en la opción *Data property assertions*. Se seleccionará el atributo que se va a definir, en el cuadro de la derecha se indicará el valor y en el desplegable el tipo de dato.

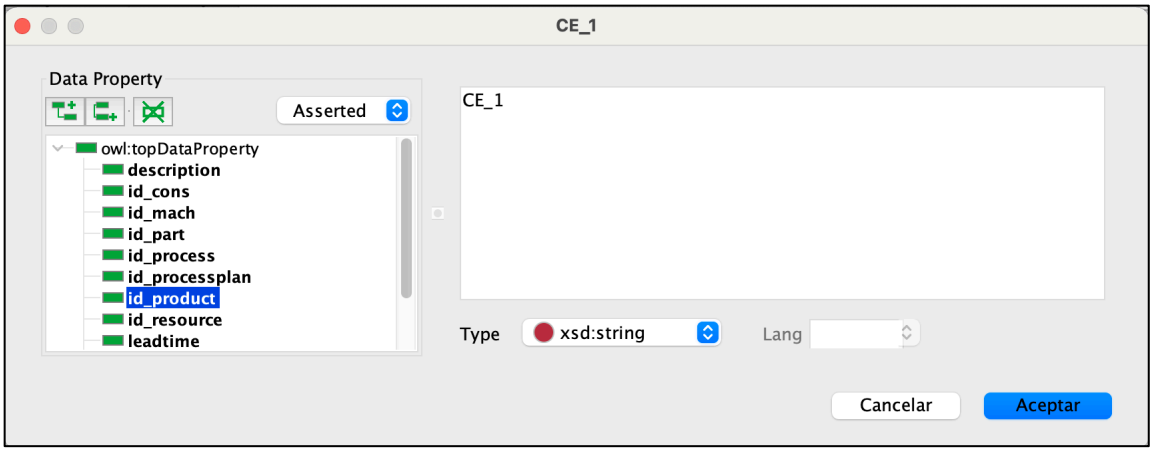


Figura 7–16. Caracterización de atributo

Las relaciones entre clases se indican en la opción *Object Property assertions*. Se establecerá el nombre de la propiedad de objeto que aplica y el nombre del individuo con el que se relaciona la instancia que se está caracterizando.

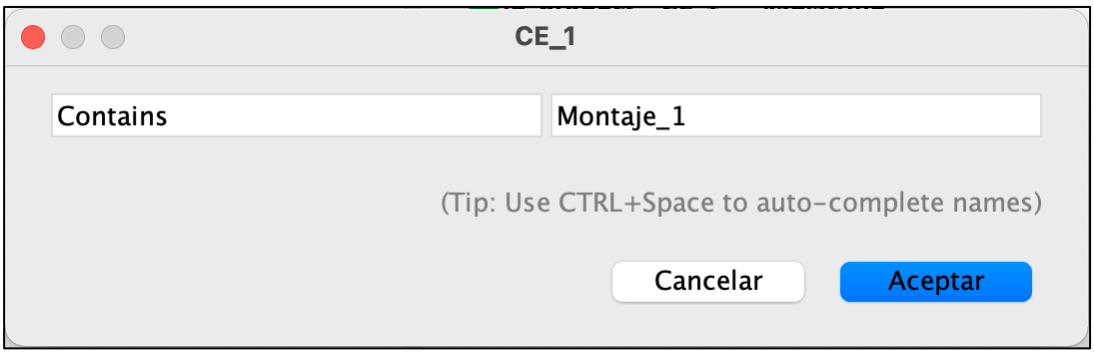


Figura 7–17. Caracterización de propiedades de objeto

En las siguientes figuras, como ejemplos, se muestran cómo deberían quedar las ventanas de propiedades de las instancias del producto final y del proceso de remachado.

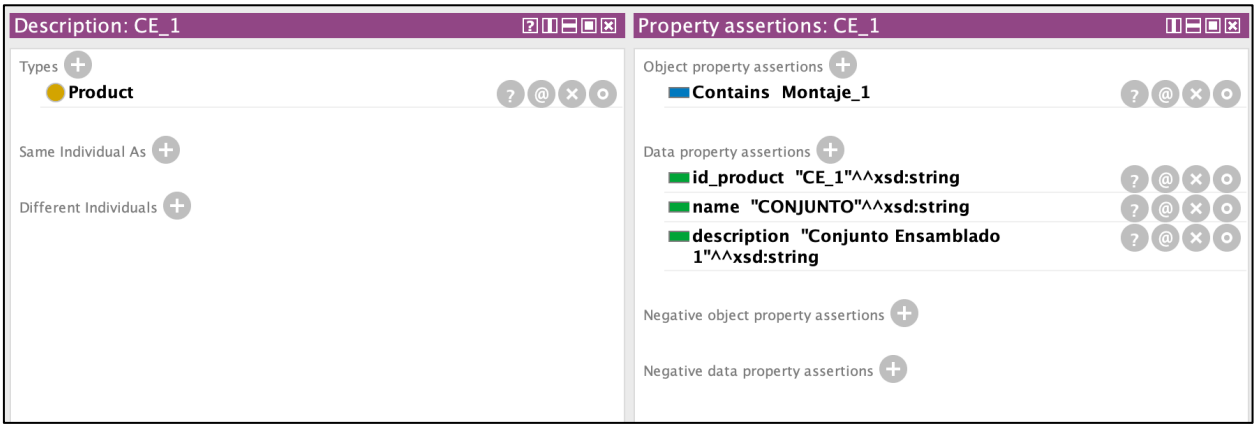


Figura 7–18. Instancia del producto final

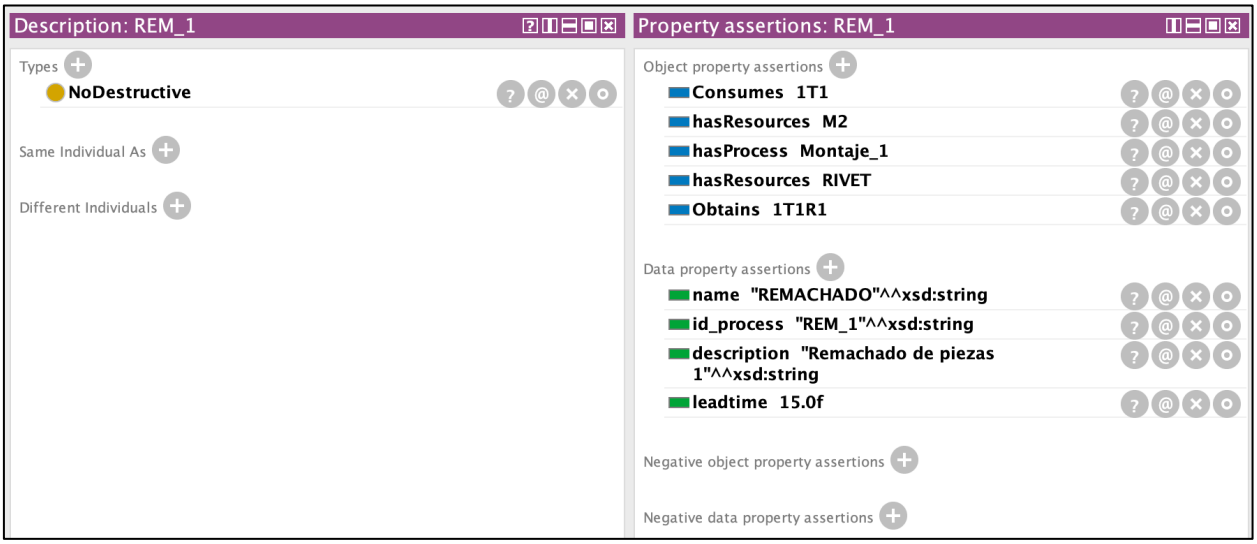
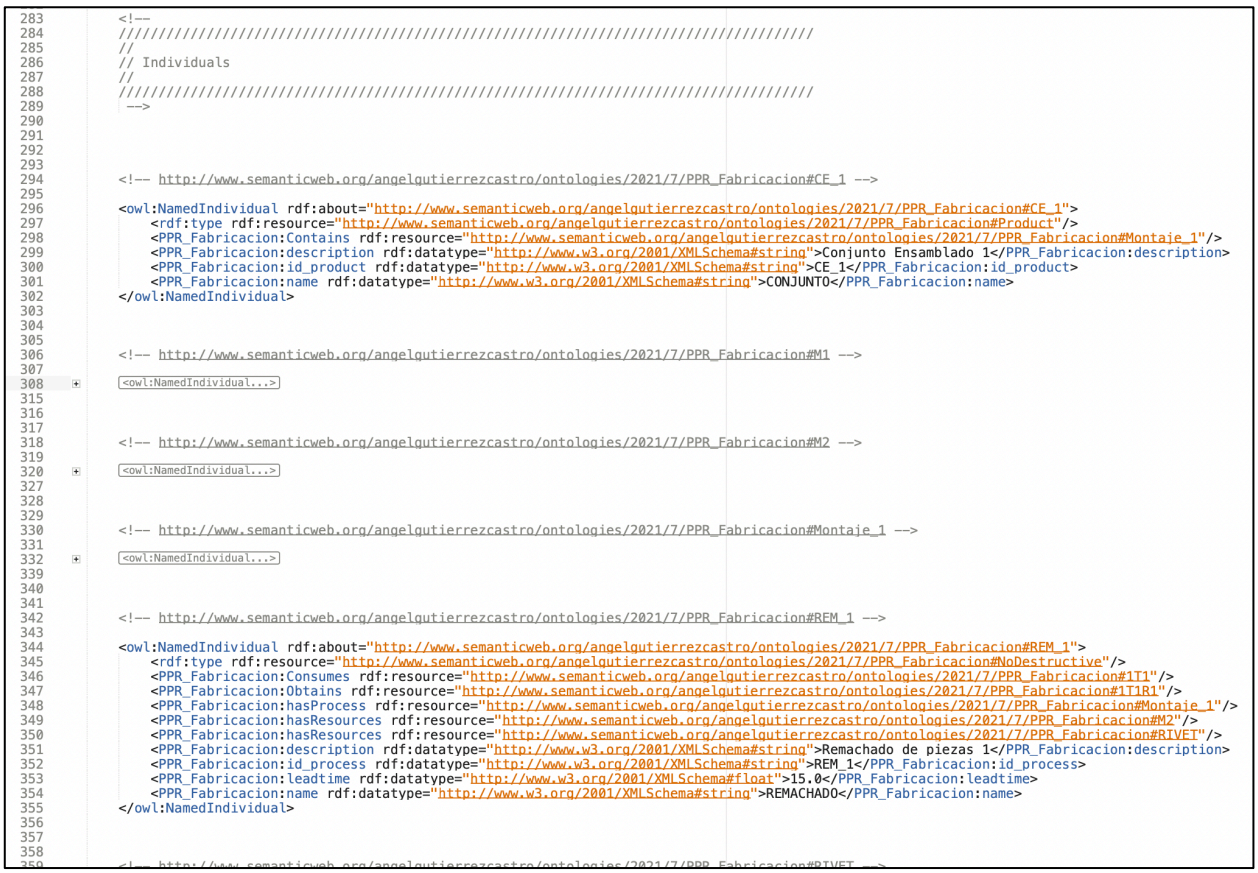


Figura 7–19. Instancia del proceso de remachado

Si se vuelve a la pestaña *Classes*, se observarán las instancias que se han definido para cada clase.

Ahora, en la exportación a XML se encontrará una nueva parte en el fichero relativa a la instanciación. Esta parte se encontrará tras la parte correspondiente a las clases.



```
    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#CE_1 -->

    <owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#CE_1">
        <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Product"/>
        <PPR_Fabricacion:Contains rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Montaje_1"/>
        <PPR_Fabricacion:description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Conjunto Ensamblado 1</PPR_Fabricacion:description>
        <PPR_Fabricacion:id_product rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CE_1</PPR_Fabricacion:id_product>
        <PPR_Fabricacion:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CONJUNTO</PPR_Fabricacion:name>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#M1 -->

    <owl:NamedIndividual...>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#M2 -->

    <owl:NamedIndividual...>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Montaje_1 -->

    <owl:NamedIndividual...>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#REM_1 -->

    <owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#REM_1">
        <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#NoDestructive"/>
        <PPR_Fabricacion:Consumes rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#1T1"/>
        <PPR_Fabricacion:Obtains rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#1T1R1"/>
        <PPR_Fabricacion:hasProcess rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#Montaje_1"/>
        <PPR_Fabricacion:hasResources rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#M2"/>
        <PPR_Fabricacion:hasResources rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#RIVET"/>
        <PPR_Fabricacion:description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Remachado de piezas 1</PPR_Fabricacion:description>
        <PPR_Fabricacion:id_process rdf:datatype="http://www.w3.org/2001/XMLSchema#string">REM_1</PPR_Fabricacion:id_process>
        <PPR_Fabricacion:leadtime rdf:datatype="http://www.w3.org/2001/XMLSchema#float">15.0</PPR_Fabricacion:leadtime>
        <PPR_Fabricacion:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">REMACHADO</PPR_Fabricacion:name>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/7/PPR_Fabricacion#RIVET -->
```

Figura 7–20. Instancias en XML

7.3 Definición de la ontología del modelo de simulación

Las ontologías no solo son usadas como medio para clasificar y organizar conceptos, éstas también sirven para construir modelos de simulación. Las ontologías asociadas a modelos de simulación reflejan técnicas de modelado, los componentes que las construyen y las reglas que definen cómo deben operar estos componentes.

El uso de ontologías garantiza la existencia de un vocabulario común y permitirá la interoperabilidad entre modelos. Por ello, los elementos de la estructura PPR construidos en base a una estructura ontológica, permitirá identificar los componentes necesarios para construir la ontología del modelo de simulación. Por ejemplo, los productos de una estructura PPR corresponderán a las entidades del modelo de simulación; o los tiempos de proceso corresponden a la duración de actividades. Será un sistema PLM el que proporcione todos los datos necesarios para construir el modelo de simulación y poder ejecutarlo.

En la siguiente figura se muestra la representación gráfica y fundamental de la ontología de modelos de simulación:

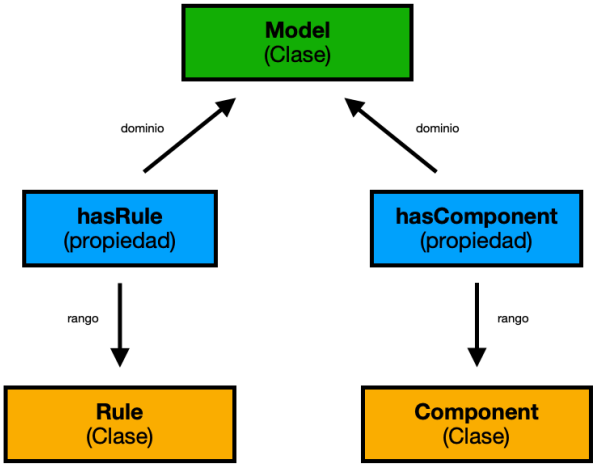


Figura 7–21. Arquitectura ontología modelo de simulación

La clase modelo corresponde a técnicas de modelado, las cuales están formadas por componentes y reglas. La clase componente hace referencia a todos los elementos a partir de los cuales se construye un modelo de simulación. Se pueden identificar dos tipos de componentes en un modelo de simulación: procesos y recursos. Los procesos son aquellas acciones que hacen cambiar el estado de un sistema y los recursos son aquellos elementos que no puede realizar acciones directamente pero que son necesarios para que los procesos lleven a cabo determinadas acciones. Así, la "superclase" `Component` podría estructurarse como se muestra en la siguiente forma:

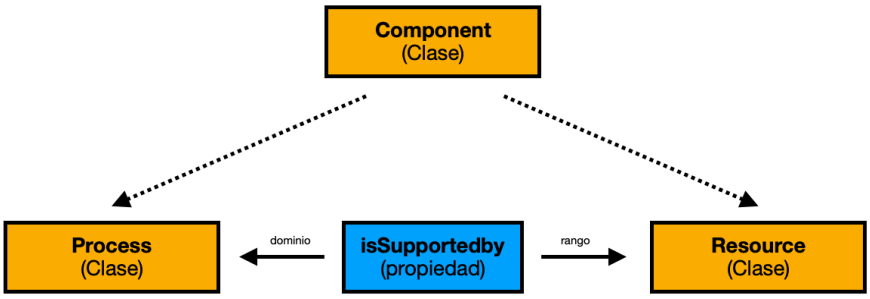


Figura 7–22. Arquitectura clase componente

Por otro lado, la clase regla establecerá cómo operan y trabajan los componentes. Esta clase estará formada por distintos tipos de mecanismos, que podrían identificarse como: activación, habilitación y programación. Las reglas de activación hacen referencia a las acciones que conllevan la puesta en marcha de procesos y recursos. Las reglas de habilitación representan las relaciones y conexiones entre eventos, es decir, las transacciones. Por ejemplo, en la técnica de modelado mediante grafos de eventos los arcos pueden ser caracterizados como reglas de activación. Por último, las reglas de programación hacen referencia a aquellas funciones, variables o parámetros necesarios para las interacciones entre componentes, estarán relacionadas con las reglas de activación. La "superclase" `Regla` quedaría construida de la siguiente forma:

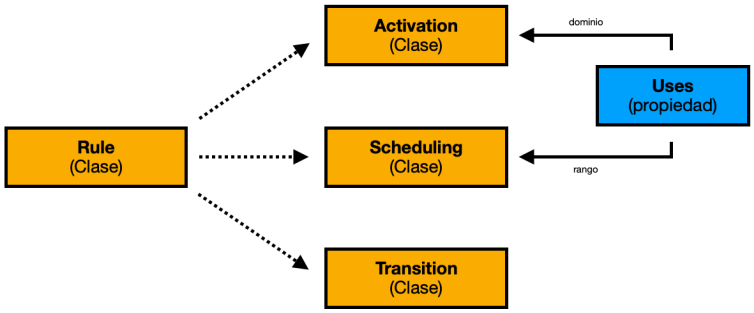


Figura 7–23. Arquitectura clase regla

Las clases de componentes y reglas habrán de estar relacionadas para que una técnica de modelado quede totalmente definida. Por ello, habrán de relacionarse las clases que forman las clases componente y reglas. A su vez, estas clases se relacionarán con la clase modelo, que representa las técnicas de modelado. De esta forma, la arquitectura de la ontología de modelos de simulación quedaría diseñada como se muestra en la figura de a continuación:

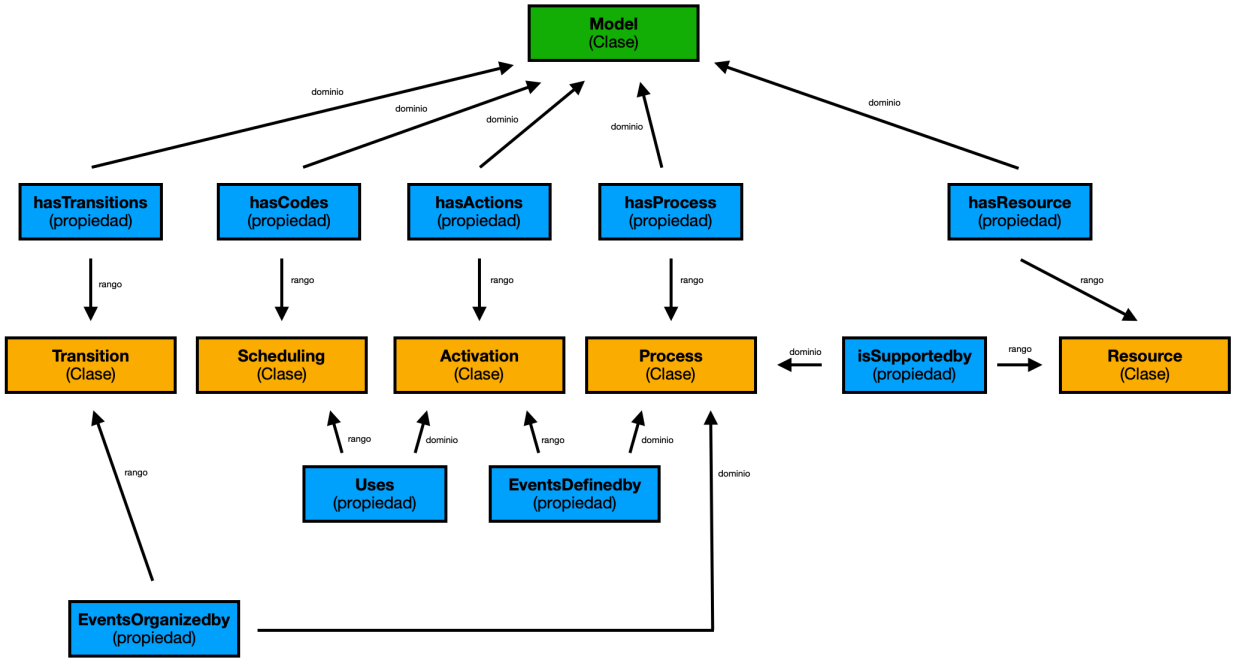


Figura 7–24. Arquitectura completa ontología modelo de simulación

En este proyecto la simulación se realizará a través del software AnyLogic. Por ello, la ontología del modelo de simulación que se acaba de desarrollar habrá de caracterizarse para este software. En el apartado 6.3 "Modelado mediante AnyLogic" de este trabajo se presentaron distintos módulos de AnyLogic necesarios para la construcción de un modelo de simulación. A su vez, se fueron conociendo las correspondencias más importantes entre el fichero XML del modelo y el diagrama de flujo; esto proporcionará la semántica que se habrá de emplear en la construcción de la ontología para que posteriormente, cuando se instancie y se exporte a un fichero XML, éste se transforme en otro fichero XML que se pueda importar en el software AnyLogic.

Las clases definidas en la ontología anteriormente definida tienen su correspondiente caracterización en AnyLogic. Por ejemplo, la clase `Process` hace referencia a los *blocks* de AnyLogic como pueden ser *Source*, *Delay*, *Service*, etc; la clase `Transition` corresponde a los arcos que unen los *blocks* del modelo, llamados *Connectors*; la clase `Scheduling` hace referencia a los componentes de AnyLogic *Variable*, *Function*, *Parameter*, etc. Además, esta caracterización tiene una correspondencia y semántica determinada en el fichero XML del modelo de simulación en AnyLogic, que será la que usará en la construcción de la ontología del modelo de simulación. En la siguiente tabla se pueden observar las correspondencias entre la ontología del modelo de simulación anteriormente definida y el fichero XML que habrá de importarse en el software AnyLogic:

Tabla 7–3. Relaciones semánticas ontología simulación y fichero XML de AnyLogic

Ontología	Fichero XML
Process	EmbbededObject
Resource	EmbbededObject
Transition	Connector
Scheduling	Variable

Activation	Parameter

A continuación, se construirá la ontología del modelo de simulación caracterizada para AnyLogic y que el modelo pueda ser ejecutado en este software. El fichero XML de AnyLogic tiene una serie de código y etiquetas en el mismo que son comunes a todos los modelos elaborados en AnyLogic que no han sido objeto de estudio en este proyecto. Por otro lado, AnyLogic cuenta con una amplia librería y opciones de modelado; no todos los componentes de AnyLogic serán definidos en la ontología de simulación. La ontología aquí diseñada contemplará aquellos módulos de modelado necesarios para la prueba de la metodología estudiada en este proyecto y con vistas a las necesidades en la instanciación que se realizará más adelante. No obstante, en el apartado 6.3 de este trabajo se encuentran estudiados más elementos de AnyLogic de los que incluirá la ontología de simulación, pero que pueden ser incluidos de semejante forma.

La arquitectura de la ontología de modelos de simulación para después ejecutar en AnyLogic se muestra a continuación:

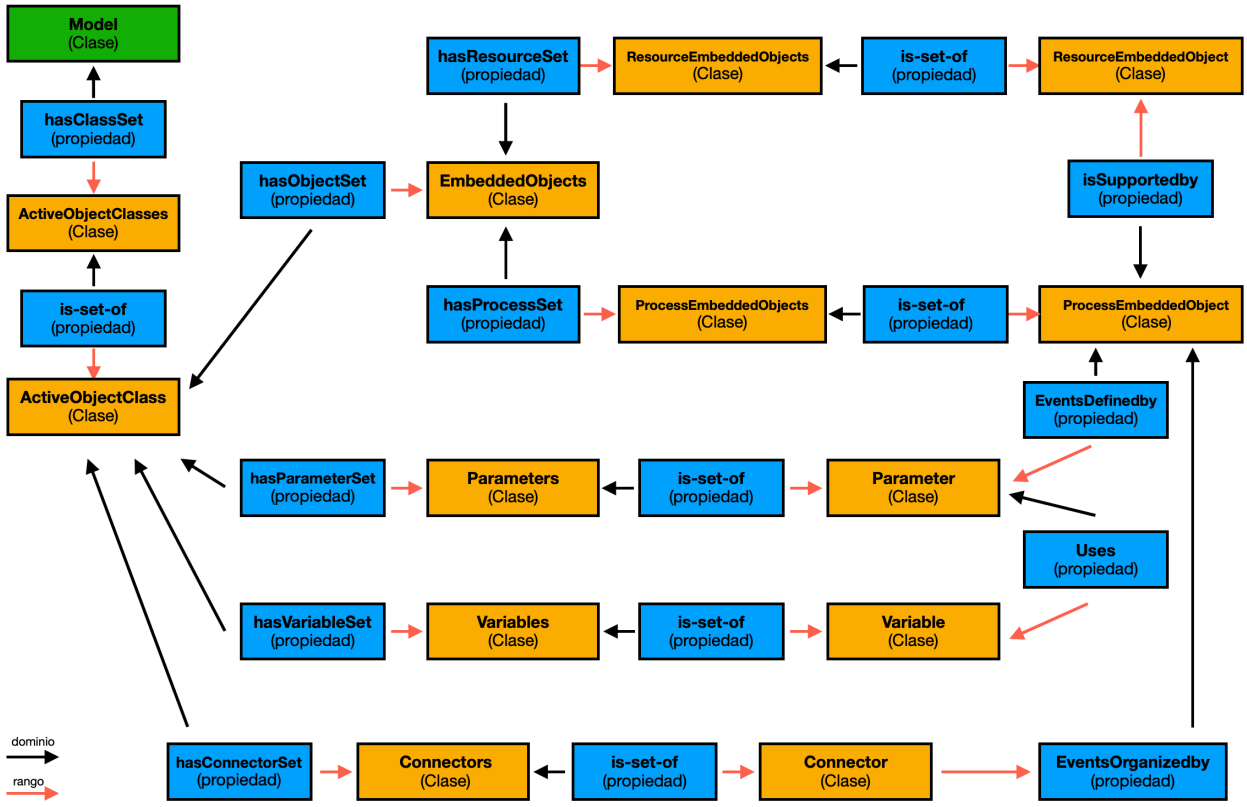


Figura 7–25. Arquitectura ontología modelo simulación AnyLogic

Nótese que se han incluido las clases `ActiveObjectClasses` y `ActiveObjectClass`, las cuales incluirán los agentes del modelo. Los agentes representan diversos elementos: unidades de equipo, productos, vehículos, ideas, proyectos…

En el diagrama anterior, por simplificación, se han incluido solo las clases generales de la ontología del modelo de simulación. No obstante, estas clases generales tienen más propiedades de objeto con otras clases, específicas ellas. A continuación, se presentarán las arquitecturas de dichas clases y, además, se incluirán los atributos de las clases, hasta ahora no definidos.

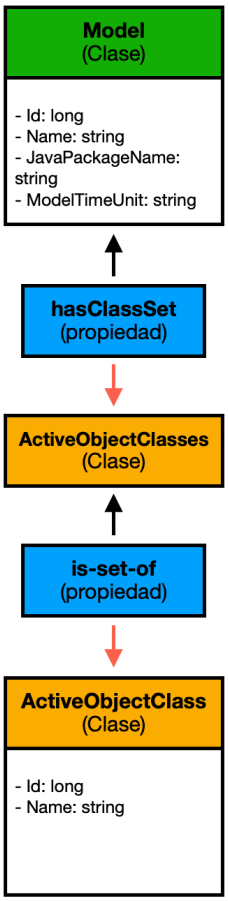


Figura 7–26. Extensión arquitectura clases Model y ActiveObjectClass

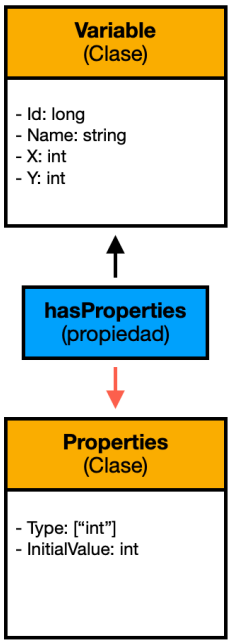


Figura 7–27. Extensión arquitectura clase Variable

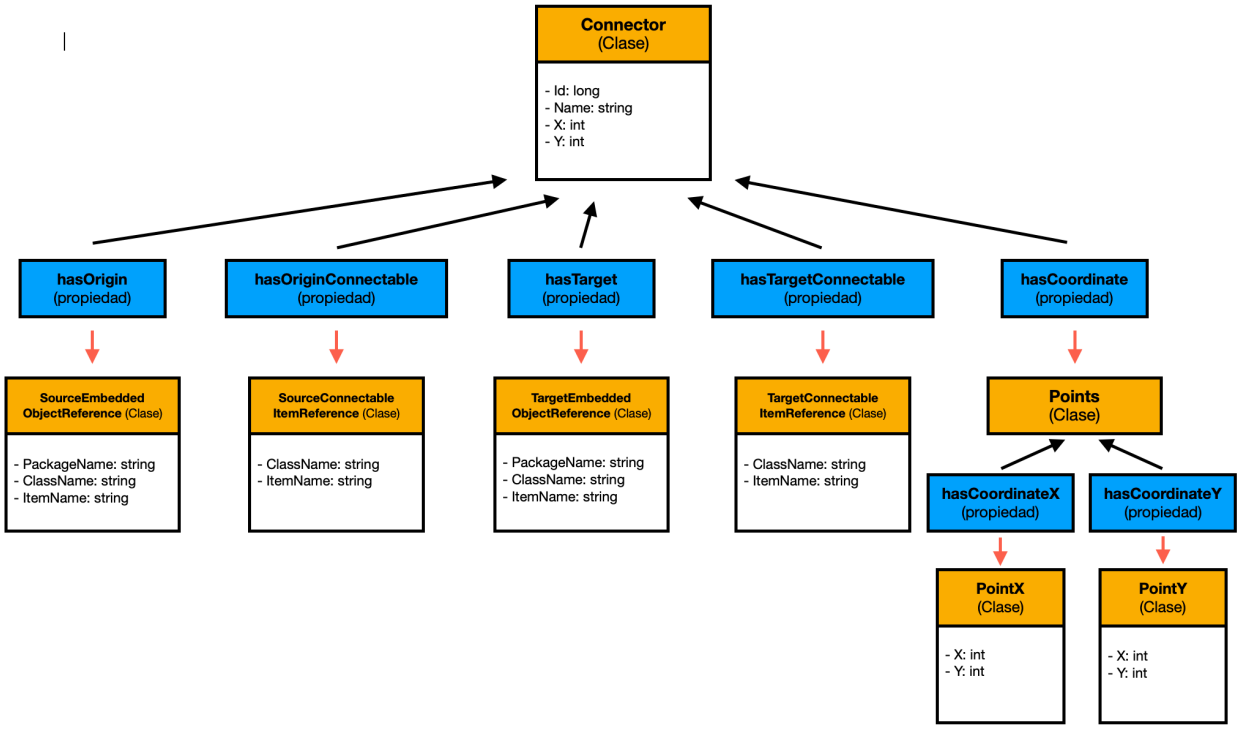


Figura 7–28. Extensión arquitectura clase Connector

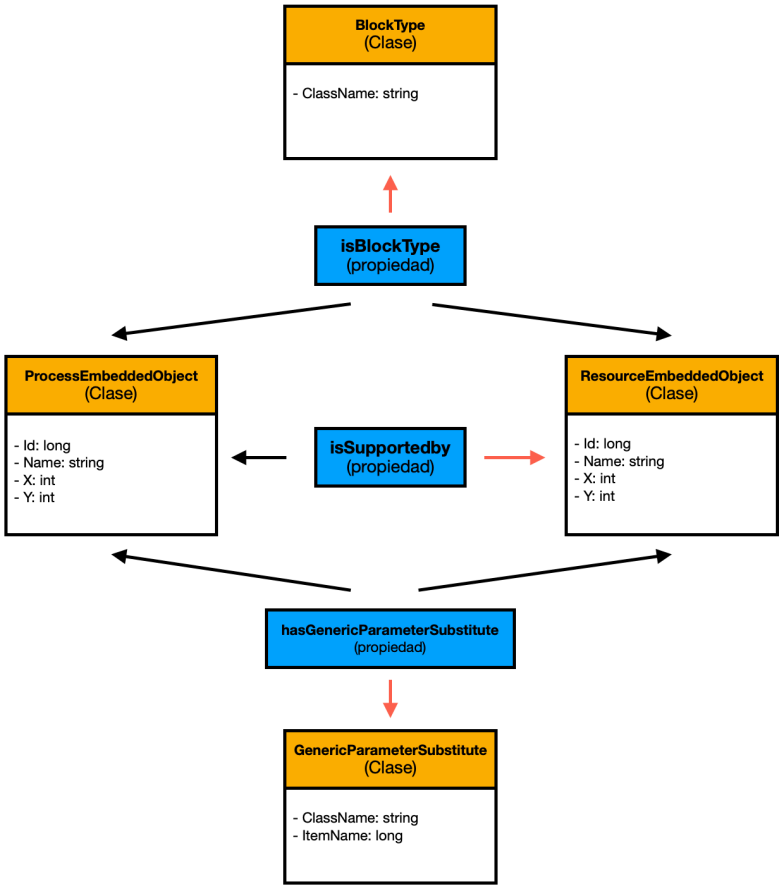


Figura 7–29. Extensión arquitectura clases EmbeddedObject

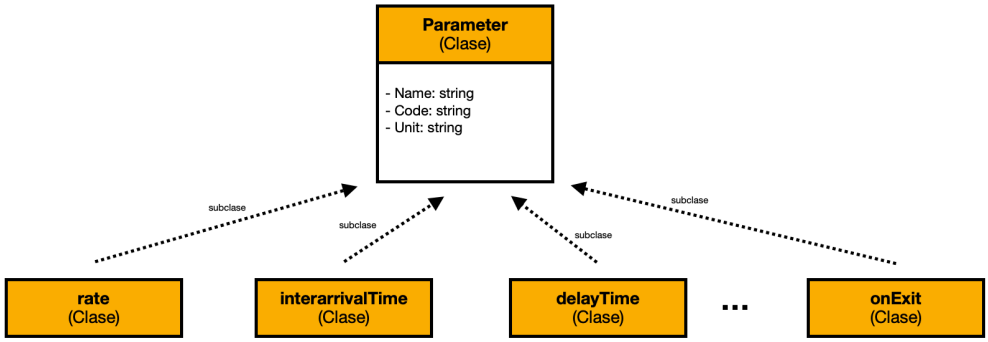


Figura 7–30. Extensión arquitectura clase Parameter

La clase `Parameter` está compuesta de subclases que definen todas las propiedades posibles que puede tener un block de AnyLogic.

La ontología aquí diseñada ha sido construida en el software Protégé para después poder instanciarla a través del mismo software y generar su fichero XML, que será del que se partirá para construir el fichero XML que deberá importar AnyLogic para ejecutar el modelo de simulación. A continuación, se muestran algunos pantallazos del software Protégé con la ontología construida:

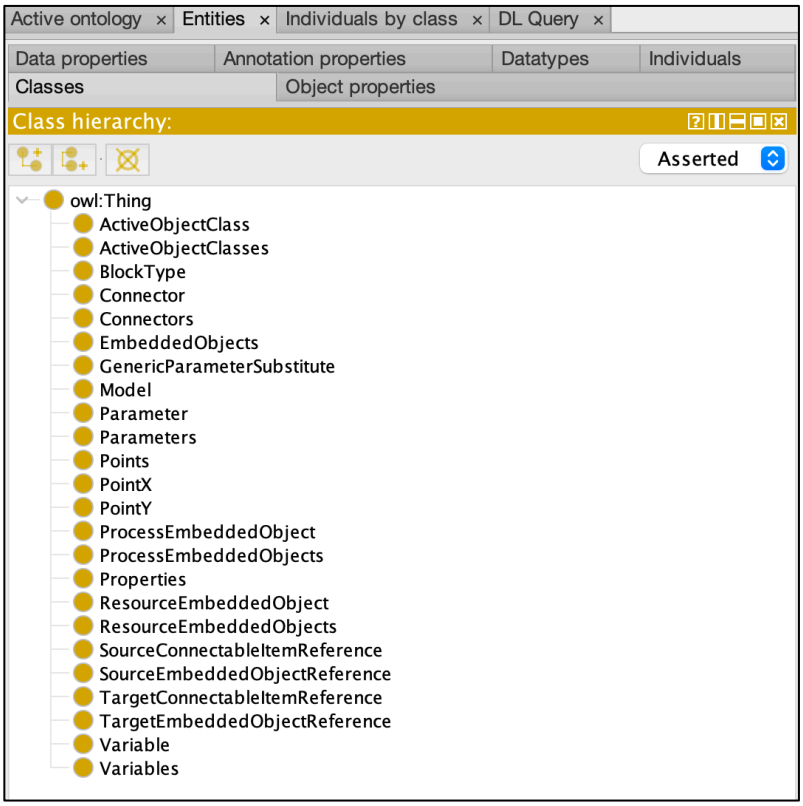


Figura 7–31. Clases ontología simulación en Protégé

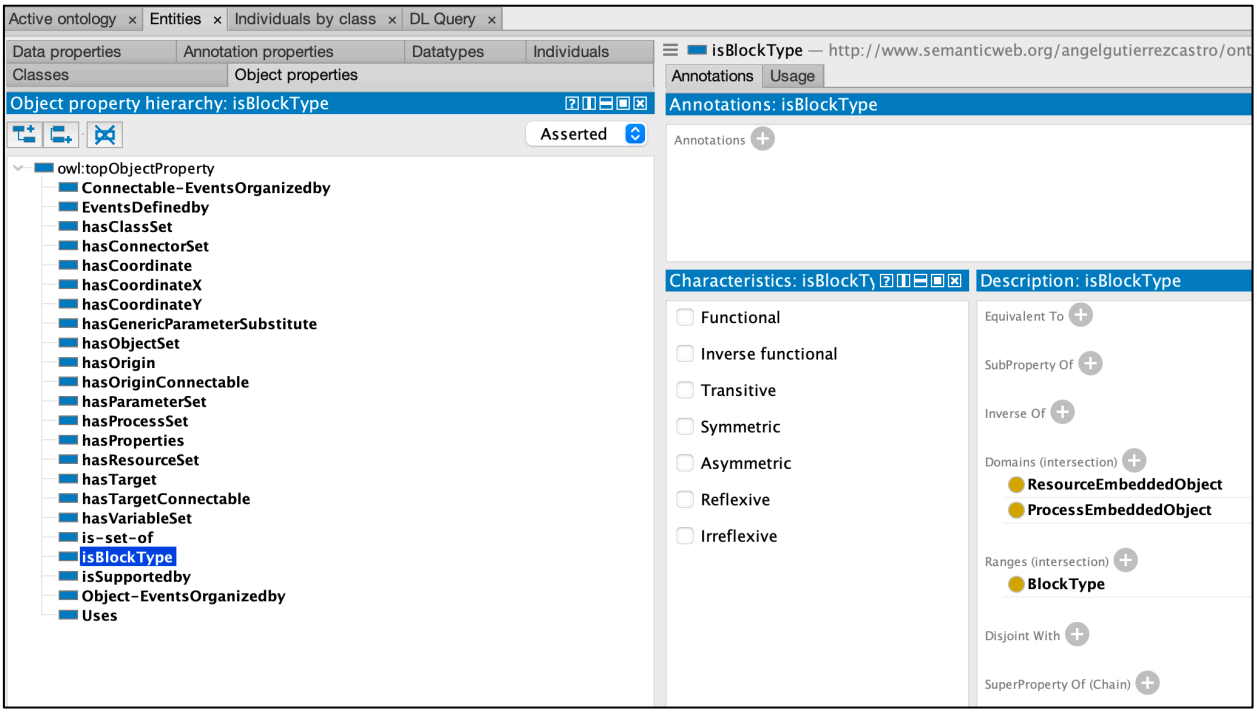


Figura 7–32. Propiedades de objeto ontología simulación en Protégé

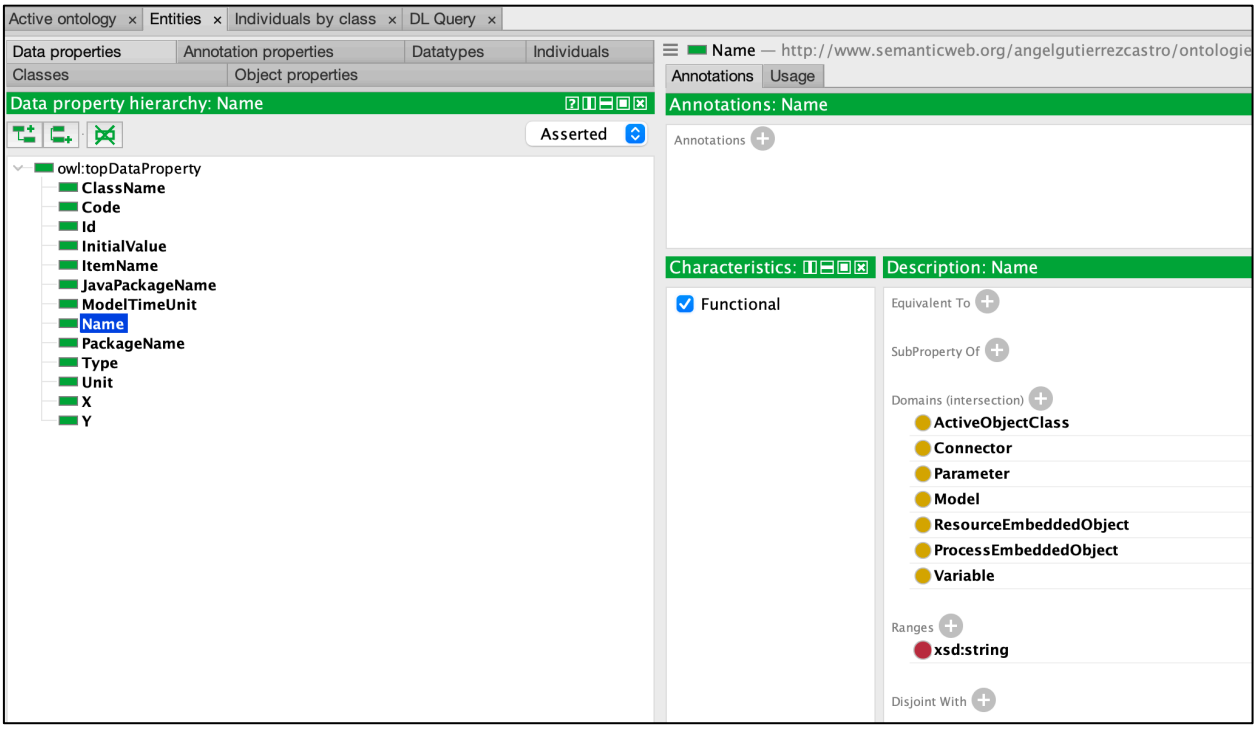


Figura 7–33. Atributos ontología simulación en Protégé

7.4 Instanciación de la ontología del modelo de simulación

El modelado de fabricación y el modelado de simulación se han desarrollado en entornos ontológicos para que estos modelos resulten ser genéricos y unívocos. Estas ontologías contemplan la organización de los conceptos y conocimientos de los modelos de fabricación y simulación, se tratan de una herramienta que proporciona descripciones exhaustivas y rigurosas para que la instanciación resulte ser codificada de forma correcta, sin errores y pérdida de conceptos e información.

El modelo de fabricación y el modelo de simulación deben operar conjuntamente y el uso de ontologías facilita la interoperabilidad entre modelos necesaria para ello. En este sentido, habrán de establecerse los vínculos necesarios entre los conceptos de ambos modelos para conseguir la interoperabilidad requerida. Para representar la visión integral de todos los datos de los modelos de fabricación y simulación, será necesario implementarlos en un sistema PLM. Así, este sistema almacenará todos los datos necesarios que establecerán los vínculos y relaciones entre ambos modelos. Resultará un modelo integrado que también quedaría definido en un entorno ontológico. Asimismo, también habrán de almacenarse las correspondencias entre el modelo de simulación y los elementos del software AnyLogic para que la instanciación del modelo pueda ejecutarse, como son todas las propiedades posibles que puede tener un block de AnyLogic.

En este apartado se realizará la instanciación de la ontología del modelo de simulación partiendo del ejemplo que se tomó en la instanciación de la ontología de la estructura PPR. Para realizar esta instanciación es necesario establecer las relaciones y vínculos entre modelos que en el párrafo anterior se mencionaba. A continuación, se elaborará la instanciación y a su vez se irán desarrollando estas relaciones y vínculos.

Se supone una línea de ensamblado en la que se obtiene un tipo de conjunto tras un plan de procesos que incluye un torneado y un remachado. Las piezas llegan al sistema cada 2 minutos y llegarán un total de 50 piezas. Seguidamente, pasarán a la cola de espera de la estación de taladrado, este búfer tendrá capacidad infinita. En la estación de taladrado las piezas serán procesadas por un taladro en un tiempo de 5 minutos. A continuación, las piezas pasarán a una estación de remachado con capacidad para una sola pieza y donde será necesario una remachadora y 20 remaches por pieza. La estación de remachado también contará con un búfer de capacidad infinita. El stock de remaches será de 500 unidades. Tras el remachado, saldrá el conjunto de la línea de montaje.

Para realizar la instanciación, en primer lugar, se habrá de abrir un nuevo proyecto en AnyLogic y exportarlo a XML, pues se crearán una serie de códigos que serán necesarios.

Se comenzará la instanciación por las clases `Model` y `ActiveObjectClass`. En este caso, solo se cuenta con una clase `ActiveObjectClass`, que será el propio `Main` del proyecto en AnyLogic dado que no habrán de crearse más agentes.

Tabla 7–4. Instanciación clases Model y ActiveObjectClass

Model	ActiveObjectClass
SimulacionEnsamblado - Id: 1630940360989 - Name: SimulacionEnsamblado - JavaPackageName: simulacionensamblado - ModelTimeUnit: Minute	**Main** - Id: 1630940360995 - Name: Main

Los códigos necesarios a los que anteriormente se hacía referencia son al atributo `Id` de la clase `Model`. El `Id` de la clase `ActiveObjectClass` es el de la clase `Model` más cuatro (esto siempre será así).

La entrada de piezas se identifica como una clase `ProcessEmbeddedObject` correspondiente a un módulo *Source* de AnyLogic con sus respectivos parámetros. Además, estas clases conllevarán la definición de otras que heredan.

Tabla 7–5. Instanciación entrada piezas

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
EntradaPiezas - Id: 1630940431296 - Name: EntradaPiezas	**EntradaPiezas_Block** - ClassName: Source	**EntradaPiezas_Generic** - ClassName: Source - ItemName: 1412336242928

- X: 50 - Y: 110		

Tabla 7–6. Instanciación parámetros entrada piezas

Parameter	Parameter	Parameter	Parameter
Entrada_arrivalType - Name: arrivalType - Code: self.INTERARRIVAL_TIME	**Entrada_interarrivalTime** - Name: interarrivalTime - Code: 2 - Unit: MINUTE	**Entrada_limitArrivals** - Name: limitArrivals - Code: true	**Entrada_maxArrivals** - Name: maxArrivals - Code: 50

El atributo `ItemName` es específico para cada módulo de AnyLogic. En este caso, para un módulo *Source* siempre será "1412336242928".

Las piezas pasan a la cola de espera de la estación de taladrado y en la estación serán procesadas por un taladro. Así, habrá que instanciar una clase `ResourceEmbeddedObject` correspondiente al taladro y una clase `ProcessEmbeddedObject` correspondiente a un módulo *Service* de AnyLogic, que es el que modela la secuencia búfer, toma de un recurso y procesado.

Tabla 7–7. Instanciación taladro

ResourceEmbeddedObject	BlockType	GenericParameterSubsitute
Taladro - Id: 1630940485496 - Name: M1 - X: 150 - Y: 210	**Taladro_Block** - ClassName: ResourcePool	**Taladro_Generic** - ClassName: ResourcePool - ItemName: 1412336243135

Tabla 7–8. Instanciación parámetros taladro

Parameter	Parameter
Taladro_type - Name: type - Code: self.RESOURCE_PORTABLE	**Taladro_capacity** - Name: capacity - Code: 1

Tabla 7–9. Instanciación estación taladrado

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
Taladrado - Id: 1630940485496 - Name: TAL_1 - X: 150 - Y: 100	**Taladrado_Block** - ClassName: Service	**Taladrado_Generic** - ClassName: Service - ItemName: 1412336243141

Tabla 7–10. Instanciación parámetros taladrado

Parameter	Parameter	Parameter	Parameter
Taladrado_seize - Name: seizeFromOnePool - Code: true	**Taladrado_resourcePool** - Name: resourcePool - Code: M1	**Taladrado_delay** - Name: delayTime - Code: 5 - Unit: MINUTE	**Taladrado_maxCapacity** - Name: maximumCapacity - Code: true

A continuación, las piezas pasarían a una estación de remachado donde se detecta que son necesarios dos recursos: la remachadora y los remaches. Dado que la estación de taladrado habrá de solicitar dos recursos diferentes, no se podrá modelar mediante un block *Service*, sino que la secuencia tendría que ser la siguiente: dos *Seize*, *Delay* y *Release*. Por otro lado, dado que el stock de remaches es consumible, entrará en juego un variable que vaya actualizando dicho stock.

Tabla 7–11. Instanciación remachadora

ResourceEmbeddedObject	BlockType	GenericParameterSubsitute
Remachadora - Id: 1630940605753 - Name: M2 - X: 290 - Y: 210	**Remachadora_Block** - ClassName: ResourcePool	**Remachadora_Generic** - ClassName: ResourcePool - ItemName: 1412336243135

Tabla 7–12. Instanciación parámetros remachadora

Parameter	Parameter
Remachadora_type - Name: type - Code: self.RESOURCE_PORTABLE	**Remachadora_capacity** - Name: capacity - Code: 1

Tabla 7–13. Instanciación variable stock remaches y sus propiedades

Variable	Properties
StockRemaches - Id: 1630940848428 - Name: StockRemaches - X: 270 - Y: 340	**StockRemaches_prop** - Type: int - InitialValue: 20

Tabla 7–14. Instanciación remaches

ResourceEmbeddedObject	BlockType	GenericParameterSubsitute
Remaches - Id: R - Name: RIVET - X: 290 - Y: 280	**Remaches_Block** - ClassName: ResourcePool	**Remaches_Generic** - ClassName: ResourcePool - ItemName: 1412336243135

Tabla 7–15. Instanciación parámetros remaches

Parameter	Parameter	Parameter
Remaches_type - Name: type - Code: self.RESOURCE_PORTABLE	**Remaches_capacity** - Name: capacity - Code: StockRemaches	**Remaches_destroy** - Name: destroyExcessUnits - Code: true

Tabla 7–16. Instanciación solicitud remaches

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
SolicitaRemaches - Id: 1630940658710 - Name: SolicitaRIVET - X: 240 - Y: 100	**SolicitaRemaches_Block** - ClassName: Seize	**SolicitaRemaches_Generic** - ClassName: Seize - ItemName: 1412336243147

Tabla 7–17. Instanciación parámetros solicitud remaches I/II

Parameter	Parameter	Parameter
SolRemaches_seize - Name: seizeFromOnePool - Code: true	**SolRemaches_resourcePool** - Name: resourcePool - Code: RIVET	**SolRemaches_quantity** - Name: resourceQuantity - Code: 20

Tabla 7–18. Instanciación parámetros solicitud remaches II/II

Parameter	Parameter
SolRemaches_maxCapacity - Name: maximumCapacity - Code: true	**SolRemaches_onExit** - Name: onExit - Code: StockRemaches=StockRemaches-20; RIVET.set_capactiy(StockRemaches);

Tabla 7–19. Instanciación solicitud remachadora

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
SolicitaRemachadora - Id: 1630940773950 - Name: SolicitaM2 - X: 340 - Y: 100	**SolicitaRemachadora_Block** - ClassName: Seize	**SolicitaRemachadora_Generic** - ClassName: Seize - ItemName: 1412336243147

Tabla 7–20. Instanciación parámetros solicitud remachadora

Parameter	Parameter	Parameter
SolRemachadora_seize - Name: seizeFromOnePool - Code: true	**SolRemachadora_resourcePool** - Name: resourcePool - Code: M2	**SolRemachadora_capacity** - Name: capacity - Code: 1

Tabla 7–21. Instanciación remachado

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
Remachado - Id: 1630940908548 - Name: REM_1 - X: 440 - Y: 100	**Remachado_Block** - ClassName: Delay	**Remachado_Generic** - ClassName: Delay - ItemName: 1412336242930

Tabla 7–22. Instanciación parámetros remachado

Parameter
Remachado_delayTime - Name: delayTime - Code: 15 - Unit: MINUTE

Tabla 7–23. Instanciación liberación remaches y remachadora

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
Libera_Rem - Id: 1630941347571 - Name: LiberaM2_RIVET - X: 530	**Libera_Rem_Block** - ClassName: Release	**Libera_Rem_Generic** - ClassName: Release - ItemName: 1412336243154

- Y: 100		

Tabla 7–24. Instanciación parámetros liberación remaches y remachadora

Parameter	Parameter
Libera_Rem_Mode - Name: releaseMode - Code: self.ALL_FROM_SEIZES	**Libera_Rem_Seize** - Name: seizeBlocks - Code: {SolicitaM2, SolicitaRIVET}

Tras el remachado, las piezas saldrían de la línea de fabricación.

Tabla 7–25. Instanciación salida piezas

ProcessEmbeddedObject	BlockType	GenericParameterSubsitute
SalidaPiezas - Id: 1630941402459 - Name: Salida_CE1 - X: 640 - Y: 110	**SalidaPiezas_Block** - ClassName: Sink	**SalidaPiezas_Generic** - ClassName: Sink - ItemName: 1412336242929

Por ultimo, quedaría por instanciar las conexiones entre módulos (por simplificación, a continuación, se muestra la instanciación de un conector de los seis que tendría este modelo).

Tabla 7–26. Instanciación conexión entre módulos I/II

Connector	SourceEmbedded ObjectReference	SourceConnectable ItemReference
Connector2 - Id: 1630940776157 - Name: connector2 - X: 0 - Y: 0	**Source_2** - PackageName: simulacionensamblado - ClassName: Main - ItemName: SolicitaM2	**SourceConnectable_2** - ClassName: Seize - ItemName: in

Tabla 7–27. Instanciación conexión entre módulos II/II

TargetEmbedded ObjectReference	TargetConnectable ItemReference	PointX	PointY
Target_2 - PackageName: simulacionensamblado - ClassName: Main - ItemName: SolicitaRIVET	**TargetConnectable_2** - ClassName: Seize - ItemName: out	**PointX_2** - X: 340 - Y: 120	**PointY_2** - X: 280 - Y: 120

La instanciación aquí definida ha sido implementada en la ontología construida en el software Protégé. Realizada esta instanciación, se exportará a XML para construir el fichero XML que deberá importar AnyLogic para ejecutar el modelo de simulación, como se verá en el siguiente apartado.

7.5 Simulación del modelo mediante AnyLogic

La simulación del modelo en AnyLogic se realizará importando un fichero XML en dicho software. Este fichero se generará de forma manual a partir de la instanciación de la ontología del modelo de simulación.

Como aplicación, se tomará de ejemplo la instanciación `Taladrado`.

La siguiente figura muestra la instanciación de esta operación en el software Protégé:

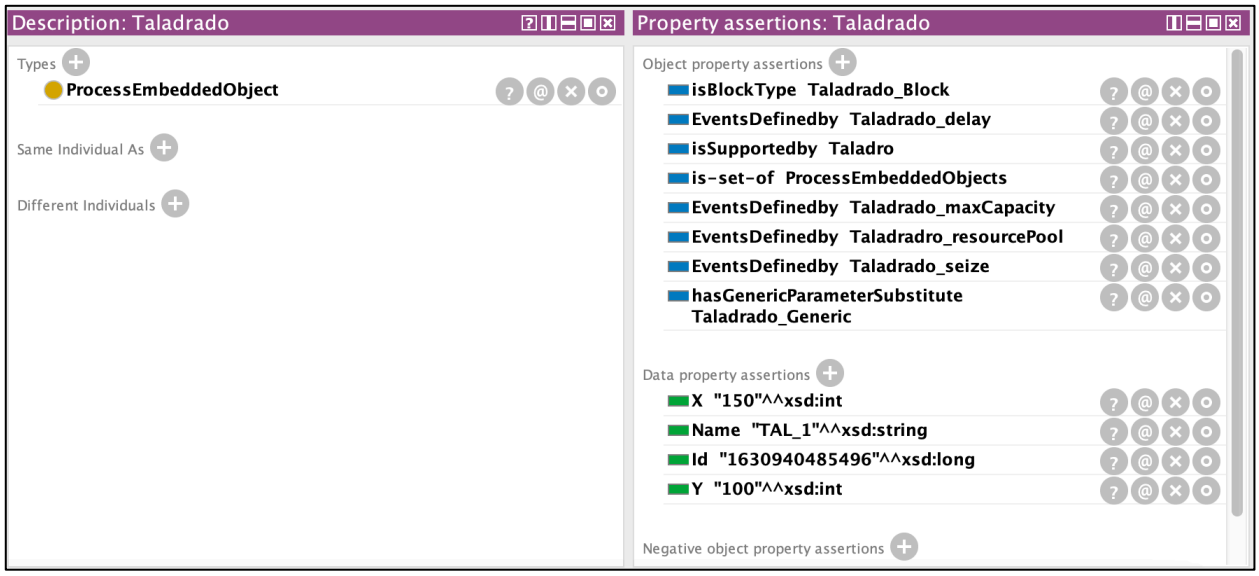


Figura 7–34. Instanciación Taladrado en Protégé

Esta instanciación en un fichero XML, junto con la instanciación de las clases heredadas, y en una primera parte, corresponde a lo siguiente:

```
<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#ProcessEmbeddedObject"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_delay"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_maxCapacity"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_seize"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladradro_resourcePool"/>
    <hasGenericParameterSubstitute rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Generic"/>
    <is-set-of rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#ProcessEmbeddedObjects"/>
    <isBlockType rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Block"/>
    <isSupportedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladro"/>
    <Id rdf:datatype="http://www.w3.org/2001/XMLSchema#long">1630940485496</Id>
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">TAL_1</Name>
    <X rdf:datatype="http://www.w3.org/2001/XMLSchema#int">150</X>
    <Y rdf:datatype="http://www.w3.org/2001/XMLSchema#int">100</Y>
</owl:NamedIndividual>


<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Block -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Block">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#BlockType"/>
    <isBlockType rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <ClassName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Service</ClassName>
</owl:NamedIndividual>


<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Generic -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_Generic">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#GenericParameterSubstitute"/>
    <hasGenericParameterSubstitute rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <ClassName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Service</ClassName>
    <ItemName rdf:datatype="http://www.w3.org/2001/XMLSchema#long">1412336243141</ItemName>
</owl:NamedIndividual>
```

Figura 7–35. Instanciación asociada al taladrado en XML I/II

En la anterior figura se han resaltado una serie de elementos del código, estos serán los datos que habrá que incluir en el fichero XML para AnyLogic.

Para pasar del fichero de la instanciación al fichero de AnyLogic, será necesario contar con los códigos vacío de cada uno de los elementos que forman un proyecto en este software. Así, en base al ejemplo que se está tratando, la instanciación de `Taladrado` se identifica como una etiqueta `EmbeddedObject` de un fichero XML de AnyLogic y se proporcionaría un código como el siguiente:

```
<EmbeddedObjects>
    <EmbeddedObject>
        <Id></Id>
        <Name></Name>
        <X></X><Y></Y>
        <Label><X>-5</X><Y>-20</Y></Label>
        <PublicFlag>false</PublicFlag>
        <PresentationFlag>true</PresentationFlag>
        <ShowLabel>true</ShowLabel>
        <ActiveObjectClass>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName></ClassName>
        </ActiveObjectClass>
        <GenericParameterSubstitute>
            <GenericParameterSubstituteReference>
                <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
                <ClassName></ClassName>
                <ItemName></ItemName>
            </GenericParameterSubstituteReference>
        </GenericParameterSubstitute>
        <Parameters>
            <Parameter>
                <Name></Name>
                <Value Class="CodeValue">
                    <Code></Code>
                    <Unit></Unit>
                </Value>
            </Parameter>
        </Parameters>
        [...]
    </EmbeddedObject>
</EmbeddedObjects>
```

Figura 7–36. Código vacío de AnyLogic para EmbeddedObject

Se observan una serie de etiquetas que no tienen contenido. Estas etiquetas tienen la misma designación de los atributos de la ontología y será aquí dónde hay que incluir los datos obtenidos de la instanciación.

```
<EmbeddedObjects>
    <EmbeddedObject>
        <Id>1630940485496</Id>
        <Name><![CDATA[TAL_1]]></Name>
        <X>150</X><Y>100</Y>
        <Label><X>-5</X><Y>-10</Y></Label>
        <PublicFlag>false</PublicFlag>
        <PresentationFlag>true</PresentationFlag>
        <ShowLabel>true</ShowLabel>
        <ActiveObjectClass>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Service]]></ClassName>
        </ActiveObjectClass>
        <GenericParameterSubstitute>
            <GenericParameterSubstituteReference>
                <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
                <ClassName><![CDATA[Service]]></ClassName>
                <ItemName><![CDATA[1412336243141]]></ItemName>
            </GenericParameterSubstituteReference>
        </GenericParameterSubstitute>
        <Parameters>
            <Parameter>
                <Name></Name>
                <Value Class="CodeValue">
                    <Code></Code>
                    <Unit></Unit>
                </Value>
            </Parameter>
        </Parameters>
        [...]
    </EmbeddedObject>
</EmbeddedObjects>
```

Figura 7–37. Instanciación asociada al taladrado en XML para AnyLogic I/II

De igual forma se procedería con la segunda parte de la instanciación correspondiente al taladrado, correspondiente de a los parámetros.

```
<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_delay -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_delay">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Parameter"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <Code rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5</Code>
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">delayTime</Name>
    <Unit rdf:datatype="http://www.w3.org/2001/XMLSchema#string">MINUTE</Unit>
</owl:NamedIndividual>


<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_maxCapacity -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_maxCapacity">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Parameter"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <Code rdf:datatype="http://www.w3.org/2001/XMLSchema#string">true</Code>
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">maximumCapacity</Name>
</owl:NamedIndividual>


<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_seize -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado_seize">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Parameter"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <Code rdf:datatype="http://www.w3.org/2001/XMLSchema#string">true</Code>
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">seizeFromOnePool</Name>
</owl:NamedIndividual>


<!-- http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladradro_resourcePool -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladradro_resourcePool">
    <rdf:type rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Parameter"/>
    <EventsDefinedby rdf:resource="http://www.semanticweb.org/angelgutierrezcastro/ontologies/2021/8/ModeloSimulacion#Taladrado"/>
    <Code rdf:datatype="http://www.w3.org/2001/XMLSchema#string">M1</Code>
    <Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">resourcePool</Name>
</owl:NamedIndividual>
```

Figura 7–38. Instanciación asociada al taladrado en XML II/II

```
<EmbeddedObjects>
    <EmbeddedObject>
        <Id>1630940485496</Id>
        <Name><![CDATA[TAL_1]]></Name>
        <X>150</X><Y>100</Y>
        <Label><X>-5</X><Y>-10</Y></Label>
        <PublicFlag>false</PublicFlag>
        <PresentationFlag>true</PresentationFlag>
        <ShowLabel>true</ShowLabel>
        <ActiveObjectClass>
            <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
            <ClassName><![CDATA[Service]]></ClassName>
        </ActiveObjectClass>
        <GenericParameterSubstitute>
            <GenericParameterSubstituteReference>
                <PackageName><![CDATA[com.anylogic.libraries.processmodeling]]></PackageName>
                <ClassName><![CDATA[Service]]></ClassName>
                <ItemName><![CDATA[1412336243141]]></ItemName>
            </GenericParameterSubstituteReference>
        </GenericParameterSubstitute>
        <Parameters>
            <Parameter>
                <Name><![CDATA[seizeFromOnePool]]></Name>
                <Value Class="CodeValue">
                    <Code><![CDATA[true]]></Code>
                </Value>
            </Parameter>
            <Parameter>
                <Name><![CDATA[resourcePool]]></Name>
                <Value Class="CodeValue">
                    <Code><![CDATA[M1]]></Code>
                </Value>
            </Parameter>
            <Parameter>
                <Name><![CDATA[maximumCapacity]]></Name>
                <Value Class="CodeValue">
                    <Code><![CDATA[true]]></Code>
                </Value>
            </Parameter>
            <Parameter>
                <Name><![CDATA[delayTime]]></Name>
                <Value Class="CodeUnitValue">
                    <Code><![CDATA[5]]></Code>
                    <Unit Class="TimeUnits"><![CDATA[MINUTE]]></Unit>
                </Value>
            </Parameter>
        </Parameters>
        [...]
    </EmbeddedObject>
</EmbeddedObjects>
```

Figura 7–39. Instanciación asociada al taladrado en XML para AnyLogic II/II

Tras realizar el mismo procedimiento para toda la instanciación, se conseguirá construir el fichero XML completo para ejecutar el modelo en el software. En las siguientes figuras se pueden observar el diagrama del modelo y su simulación.

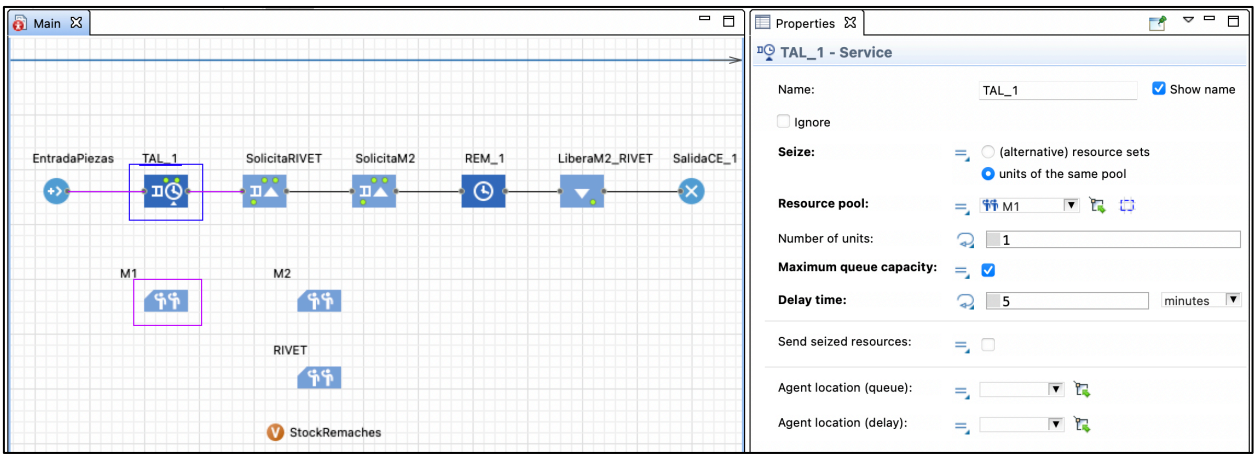


Figura 7–40. Modelo construido en AnyLogic

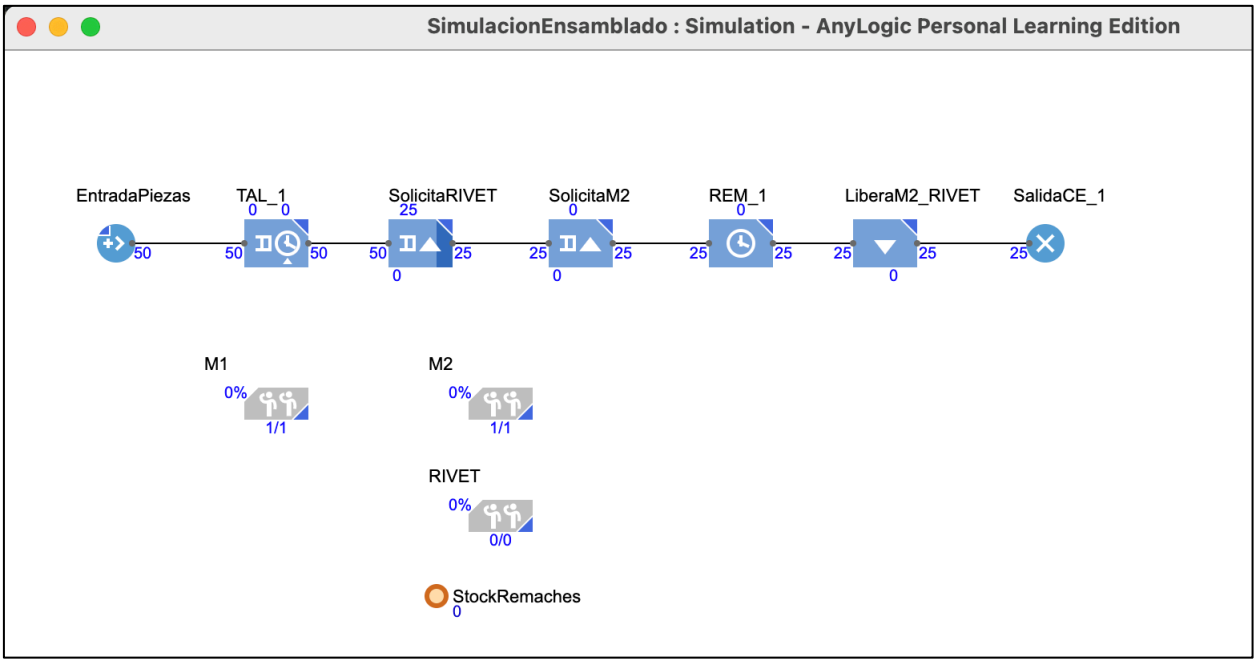


Figura 7–41. Simulación del modelo

8 CONCLUSIONES

La elevada complejidad de los sistemas industriales actuales hace necesario el uso de metodologías como MBSE, una herramienta que facilita la ingeniería de sistemas. El hecho de centrar el diseño de un sistema en modelos, en lugar de documentos, permite la detección de problemas de diseño con antelación y facilita la colaboración entre las partes interesadas.

En este mismo proyecto, donde se ha llevado a cabo la demostración mediante una simple línea de ensamblado, se ha comprobado que el uso de metodologías MBSE es necesario y beneficioso; ha sido constante la aplicación de las actividades en las que se sustenta MBSE (identificación de requisitos, análisis, pruebas, etc.).

El uso de modelos resulta ventajoso en el desarrollo de simulaciones y experimentos virtuales, evitando de esta forma los experimentos físicos, como son los prototipos. El enfoque MBSE tiene dos conceptos muy presentes en su metodología: verificación y validación. Por ello, los ensayos del sistema mediante simulaciones toman gran relevancia.

Este proyecto ha tenido como objetivo el diseño, desarrollo y prueba de una metodología para la simulación de sistemas de fabricación, basándose en la metodología MBSE.

El prototipo desarrollado en este proyecto para elaborar el modelado de simulación se ha llevado a cabo en un entorno ontológico. Esto ha permitido establecer unos conceptos y unas relaciones claras del modelo, que ha hecho que la instanciación llevada a cabo se haya podido ejecutar de forma satisfactoria.

En relación con lo anterior, se podría comentar que el software Protégé, con el que se han desarrollado las ontologías e instanciaciones, es muy intuitivo y cómodo de usar, aunque no permite la definición de una ontología de forma gráfica, lo que quizá hubiera sido más ventajoso dado el número de clases, atributos y relaciones que se han manejado.

Las ontologías son unas herramientas necesarias para la interoperabilidad entre sistemas. Se ha comprobado que el hecho de haber conceptualizado un sistema de fabricación y el modelo de simulación en entornos ontológicos ha permitido la identificación rápida y correcta de los elementos y conceptos para realizar la simulación de un sistema de fabricación.

Por otro lado, ha sido necesario caracterizar el modelo de simulación al software empleado para la ejecución de la simulación (AnyLogic, en este caso). Se ha desarrollado un modelo de simulación válido para técnicas de modelado, en general, pero que, al querer ejecutarlo mediante un software, se ha tenido que caracterizar para el software en cuestión, y para lo que ha sido necesario conocer en detalle el software.

Se podría concluir diciendo que para el diseño de modelos de simulación es necesario tomar metodologías bien implementadas para que los modelos resulten eficaces y eficientes. Sin embargo, a raíz de la búsqueda de información para el desarrollo de este proyecto, se ha comprobado que son escasas las metodologías desarrolladas para apoyar la simulación de los sistemas de fabricación, herramienta muy potente para la verificación y validación dentro de un sistema industrial.

Referencias

[1] R. Karban, M. Zamparelli, B. Bauvir, B. Koehler, L. Noethe y A. Balestra, «Exploring Model Based Engineering for Large Telescopes: getting started with descriptive models,» de *SPIE Astronomical Telescopes + Instrumentation*, Marseille, 2008.

[2] S. P. Frechette, «Model Enterprise for Manufacturing,» de *44th CIRP International Conference of Manufacturing Systems*, Madison, 2011.

[3] C. Neal, «Incentives White Papers for Advanced Manufacturing Technology,» 2009.

[4] J. W. Forrester, «Industrial dynamics,» *The MIT press,* 1961.

[5] O. Batarseh, L. McGinnis y J. Lorenz, «MBSE Supports Manufacturing System Design,» 2012.

[6] A. Azevedo y A. Almeida, «Factory Templates for Digital Factories Framework,» *Robotics and Computer Integrated Manufacturing,* vol. 27, pp. 755-771, 2011.

[7] T. R. Gruber, «A traslation approach to portable ontology specification,» de *Knowledge Acquisition, Vol. 5, V. 2*, 1993, pp. 199-220.

[8] F. Gandon y G. Schreiber, «RDF 1.1 XML Syntax,» W3C Recommendation, 2014. [En línea]. Available: https://www.w3.org/TR/rdf-syntax-grammar/.

[9] D. Brickley y R. Guha, «RDF Schema 1.1,» W3C Recommendation, 2014. [En línea]. Available: https://www.w3.org/TR/rdf-schema/.

[10] D. L. McGuinness y F. van Harmelen, «OWL Web Ontology Language Overview,» W3C Recommendation, 2014. [En línea]. Available: https://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.2.

[11] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau y J. Cowan, «Extensible Markup Language (XML) 1.1 (Second Edition),» W3C Recommendation, 2006. [En línea]. Available: https://www.w3.org/TR/2006/REC-xml11-20060816/.

[12] K. Agyapong-Kodua, C. Haraszkób y I. Némethb, «Recipe-based Integrated Semantic Product, Process, Resource (PPR) Digital Modeling Methodology,» *Procedia CIRP,* vol. 17, pp. 112-117, 2014.

[13] A. Madni y M. Sievers, «Model-based systems engineering: Motivation, current status and research opportunities,» *System Engineering,* vol. 21, pp. 172-190, 2018.

[14] F. Wilking, B. Schleich y S. Wartzack, «MBSE along the Value Chain - An Approach for the Compesation of Additional Effort,» de *2020 IEEE 15th International Conference of System of Systems*

Engineering (SoSE), Budapest, 2020.

[15] PivotPoint Technology Corp., «MBSEworks.com,» 2018. [En línea]. Available: https://mbseworks.com/.

[16] W. W. Royce, «Managing the Development of Large Software Systems,» *Proceedings of IEEE WESCON,* pp. 1-9, 1970.

[17] B. W. Boehm, «A Spiral Model of Software Development and Enhancement,» *Computer,* pp. 61-72, 1988.

[18] K. Forsberg y H. Mooz, «The Relationship of Systems Engineering to the Project Cycle,» *Engineering Management Journal,* vol. 4, nº 3, pp. 36-43, 1992.

[19] OMG, «OMG Unified Modeling Language (OMG UML), Superstructure,» 2011. [En línea]. Available: https://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.

[20] S. Friedenthal y R. Burkhart, «Evolving SysML and the System Modeling Enviroment to Support MBSE,» *INSIGHT,* vol. 18, pp. 39-41, 2015.

[21] OMG, «OMG System Modeling Language (OMG SysML),» 2019. [En línea]. Available: https://www.omg.org/spec/SysML/1.6/PDF.

[22] *IBM Harmony Deskbook 4.0.* [Art]. IBM.

[23] J. A. Estefan, «Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev. B,» INCOSE MBSE Focus Group, 2008.

[24] INCOSE, «Object-Oriented SE Method,» [En línea]. Available: https://www.incose.org/incose-member-resources/working-groups/transformational/object-oriented-se-method.

[25] D. A. Wagner, M. B. Bennett, R. Karban, N. Rouquette, S. Jenkins y M. Ingham, «An ontology for State Analysis: Formalizing the mapping to SysML,» de *2012 IEEE Aerospace Conference*, 2012.

[26] L. W. Schruben, «Simulation Modeling with Event Graphs,» *Communications of the ACM,* vol. 26, pp. 957-963, 1983.

[27] G. A. Silver, J. A. Miller, M. Hybinette, G. Baramidze y W. S. York, «DeMO: An Ontology for Discrete-event Modeling and Simulation,» *Simulation,* vol. 87, nº 9, pp. 747-773, 2011.

[28] F. Ameri y D. Dutta, «Product Lifecycle Management: Closing the Knowledge Loops,» *Computer-Aided Design and Applications,* vol. 2, nº 5, pp. 577-590, 2013.

[29] CIMdata, «Product lifecycle management: empowering the future of business,» 2002.

[30] Active Sensing, Inc, «Prodct Lifecycle Management (PLM),» 2006. [En línea]. Available: https://www.product-lifecycle-management.com/.